Resumen

En este proyecto de fin de carrera se ha llevado a cabo el diseño de unas librerías en lenguaje C para el uso eficiente de un módulo de Radio Frecuencia (RF) modelo nRF24L01 de modo que permita establecer una comunicación RF efectiva y fiable entre microcontroladores PIC18F. Concretamente, se destina este proyecto al desarrollo de asignaturas sobre el uso y programación de microcontroladores. Se obtiene tras el proceso de prueba de las comunicaciones y de validación, un sistema robusto y fácilmente programable que permite una comunicación RF rápida y fiable.

En las primeras fases del proyecto se realiza un análisis de antecedentes en el que se lleva a cabo una familiarización con la placa de desarrollo Open18f4520 de *WaveshareElectronics* y con el microcontrolador PIC18F4520 y su estándar de comunicación SPI (*Serial Peripheral Interfase*), desarrollando códigos en lenguaje C para comprender su estructura y el funcionamiento de esta comunicación. Como paso previo al desarrollo de funciones propias para el control del módulo de RF nRF24L01, se analizan códigos ya existentes concluyendo en su carencia de fiabilidad y corrección.

Posteriormente se crean las primeras funciones para comunicar el nRF24L01 con el PIC18 vía SPI, mediante las cuales se hace posible el envío de configuraciones al módulo RF así como la lectura del estado del proceso por parte del PIC18. A partir de estas funciones se hace posible implementar la inicialización de un módulo RF como transmisor (TX) o como receptor (RX) y cargar en él la configuración que desee el estudiante.

Para terminar la librería, se diseñan las funciones de envío y recepción de un paquete de datos que a la vez retornan al código principal información del proceso de comunicación, informando del correcto envío/recepción o del error acontecido. Así mismo se crean las funciones de apagado del módulo RF y de la desactivación de la comunicación por SPI.

Respecto al método de trabajo, en cada fase de desarrollo se pasa a un nivel de abstracción mayor, verificándose en cada paso el correcto funcionamiento del sistema. Se explican también los problemas y errores surgidos, el método de identificación de sus causas, y las mejoras o soluciones propuestas e implementadas.

Finalmente, tras la fase de test y validación del proceso, se concluye la corrección de la librería elaborada, que permite establecer de forma sencilla comunicaciones rápidas, fiables y totalmente controladas entre microcontroladores PIC18.

Índice

Anexos (I)

Anexos (II)

1. Glosario

- BF: *MSSP Buffer Full Status bit*
- CE: *nRF24L01 Chip Enable*
- CPHA: *Clock Phase*. Para entornos de Microchip: CKE=noCPHA.
- CPOL: *Clock Polarity*. Para entornos de Microchip: CKP=CPOL.
- CRC: *Cyclic Redundancy Check*
- CSN: *Chip Select Not*
- FIFO: *First In First Out memory*
- IRQ: *Interrupt Request*
- ISM: *Industrial, Scientific and Medical*
- LNA: *Low Noise Amplifier*
- LSB: *Least Significant Bit*
- LSByte: *Least Significant Byte*
- MAX_RT: *Maximum number of ReTrasmission interrupt*
- Mbps: *Megabit per second*
- MISO: *Master Input, Slave Output*
- MOSI: *Master Output, Slave Input*
- MSB: *Most Significant Bit*
- MSByte: *Most Significant Byte*
- MSSP: *Master Synchronous Serial Port*
- NSS: *Not Slave Select*
- PSP: *Parallel Slave Port*
- RAEE: Residuos de Aparatos Eléctricos y Electrónicos
- RF: Radio Frecuencia
- RX: Receptor
- RX_DR: *Receive Data Ready*
- SAR: *Specific Absortion Rate*
- SCLK: *SPI Serial Clock*
- SDI: *Slave Data In*
- SDO: *Slave Data Out*

- SMP: *Data sample timing*
- SPI: *Serial Peripheral Interface*
- SSPBUF: *MSSP Serial Input Buffer*
- SSPEN: *Master Synchronous Serial Port Enable bit*
- SSPIE: *MSSP Interrupt Enable bit*
- SSPIF: *MSSP Interrupt Flag bit*
- SSPIP: *MSSP Interrupt Priority bit*
- SSPSR: *MSSP Shift Register*
- TX: Transmisor
- TX_DS: *Transmit Data Sent*

2. Introducción

El objetivo de este proyecto es crear unas librerías en lenguaje C para el uso eficiente de un módulo de Radio Frecuencia modelo nRF24L01 y así establecer comunicaciones RF efectivas y fiables entre microcontroladores PIC18F.

Este proyecto nace de la iniciativa del Departamento de Ingeniería Electrónica de la ETSEIB UPC de desarrollar asignaturas sobre el uso y programación de microcontroladores; de modo que, mediante estas librerías y unos conocimientos iniciales sobre programación, sea fácil y efectivo comunicar las placas de los estudiantes entre sí. Concretamente, las próximas asignaturas que incluirán prácticas de Radio Frecuencia usando estas librerías serán:

- 240607: Desarrollo de aplicaciones basadas en microcontroladores (OPTATIVA)
- Ampliación de Electrónica (MÁSTER)

2.1. Objetivos del proyecto

- Establecer una comunicación fiable y ágil entre el PIC18F4520 y el módulo nRF24L01 que permita en todo momento verificar el buen funcionamiento del último.
- Elaborar funciones en lenguaje C para que los PIC18 carguen una configuración óptima sobre los nRF24L01 e inicien y monitoricen una comunicación RF rápida y controlada entre dos o más PIC18.
- Crear un sistema de detección y actuación ante posibles errores en el funcionamiento del nRF24L01.
- Establecer un sistema de detección de errores en la transmisión de datos por RF para que se pueda verificar en todo momento el envío y recepción de información.

2.2. Alcance del proyecto

Éste es un proyecto de diseño de software que utiliza como hardware de soporte la placa Open18F4520 de *WaveshareElectronics*. Es una placa dotada de un microcontrolador PIC18F4520 de *Microchip* y de una gran variedad de periféricos, entre ellos, el módulo *NRF24L01 RF Board*, una pequeña placa fabricada por *Waveshare* a partir del chip nRF24L01 de *Nordic*.

La finalidad de este trabajo es, en definitiva, crear una herramienta de trabajo para los estudiantes de electrónica en sus prácticas de Radio Frecuencia; de modo que no se profundizará ni divagará en la gran variedad de usos prácticos y ventajas que puede suponer la comunicación inalámbrica entre microcontroladores.

En el momento en que se alcance una comunicación sencilla, versátil, eficiente y fiable entre diferentes PIC18 se dará por satisfecho el objetivo principal de este proyecto.

3. Análisis de antecedentes

En una primera fase del proyecto, antes de investigar y trabajar propiamente sobre el módulo de Radio Frecuencia nRF24L01, se han llevado a cabo algunos estudios y trabajos previos que deben ser comentados para la buena comprensión del proyecto desarrollado.

Tras el estudio y la familiarización con el entorno de trabajo (software MPLAB y compilador C18) y con las características principales del PIC18, se ha investigado sobre el periférico de comunicación SPI (*Serial Peripheral Interface*) que dispone el PIC18 y que será utilizado para la comunicación con el módulo nRF24L01. Posteriormente se han conseguido encontrar dos códigos creados anteriormente para el control del nRF24L01: el primero, elaborado por *WaveshareElectronics*, y el segundo, diseñado para microcontroladores AVR y depositado en *Google Code*. Ambos se han estudiado, valorado y tomado en cuenta para la elaboración de las librerías definitivas.

3.1. El PIC18F4520 en la placa Open18F4520, el software MPLAB y su compilador C18

Para el aprendizaje del entorno MPLAB y del uso del lenguaje C adaptado al compilador de *Microchip* C18 se han hecho uso de los manuales *MPLAB IDE User's Guide*, *MPLAB C18 C COMPILER User's Guide* y *MPLAB C18 C Compiler Libraries*.

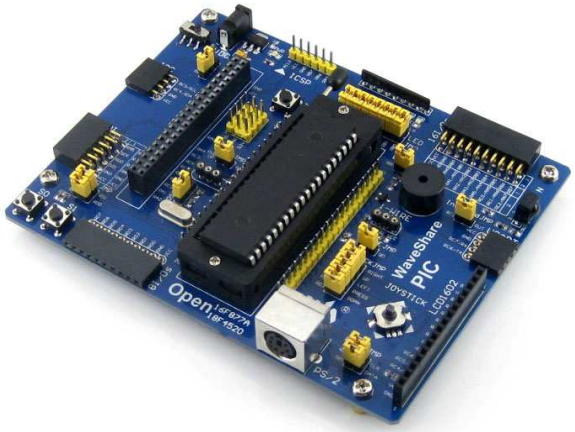


Figura 3-1: Placa de desarrollo Open18f4520

La documentación sobre la placa Open18F4520 se ha obtenido de la web de *WaveshareElectronics*, y las fuentes de información para el estudio de la estructura y características del PIC18 han sido el manual *Microcontroladores PIC: La clave del diseño* y el propio *Datasheet* del PIC18F4520, del cual se ha recogido un extracto en el **ANEXO A**. En éste se especifican algunas de las principales características y prestaciones del microcontrolador (de funcionamiento básico o relacionadas con la comunicación SPI) que además se mencionan ahora muy brevemente:

- El **PIC18F4520** es un microcontrolador de **8 bits** y 40 pines con una arquitectura de memoria Harvard: **32 KBytes** de **memoria de programa** y **1536 Bytes** de **memoria de datos**.
- La frecuencia máxima que puede soportar es de 40 MHz, aunque el **oscilador de la placa Open 18F4520 es de 4 MHz**, por lo que la frecuencia de trabajo es de 1 MHz.
- Dispone de **5 puertos de entrada/salida**: los puertos A, B, C y D son de 8 bits y el puerto E es de 4 bits.
- Dispone de Comunicaciones en Paralelo (PSP) y de Comunicaciones en Serie:
  - *Enhanced* USART (comunicación asíncrona).
  - **MSSP (*Master Synchronous Serial Port*)**, el cual puede trabajar en dos modos: I^2C y **SPI**.
- Dispone además de muchas otras características destacables como el *Watchdog Timer* o los dos niveles de interrupción de programa y también de diversos periféricos: 4 *Timers*, convertidor analógico/digital de 10 bits, comparadores, etc.

Por lo que a la **placa Open18F4520** se refiere, cabe decir que dispone de ciertos periféricos de entrada (como **pulsadores**), elementos de salida (de los cuales se han utilizado solamente los **8 LEDs** incorporados en la placa, el módulo de **7 segmentos** y la **pantalla LCD**), y además de *jumpers* de simple puenteo de estos periféricos y alguno de configuración (como el de selección de oscilador de placa o externo o el de selección de tensión de 5V o 3,3V: ajustado en todo momento a **3,3V** para no exceder la tensión máxima de alimentación de algunos periféricos como la pantalla LCD o, como se observará posteriormente, del módulo *NRF24L01 RF Board*). Se incorpora del mismo modo la información de mayor importancia de la placa de trabajo (como su esquemático general) en el **ANEXO A**.

3.2. La comunicación SPI

Una segunda fase de este proyecto se centra ahora en un elemento clave: **la comunicación entre PIC18 y nRF24L01**. Esta comunicación, como se comprobará en los siguientes apartados, permite al microcontrolador cargar una configuración determinada en el módulo RF, activar y desactivar el envío y recepción de datos, y recoger en cada momento información del proceso para poder controlarlo.

Es necesario remarcar, antes de continuar, la importancia de esta fase del trabajo, ya que algunos de problemas que han surgido en el proceso de programación del nRF24L01 han estado relacionados con ciertas limitaciones o con configuración poco eficientes o incorrecta del protocolo de comunicación.

La conexión y comunicación entre PIC18 y nRF24L01 se lleva a cabo fundamentalmente mediante **Bus SPI** (*Serial Peripheral Interface*), además de otras dos señales (como se verá más adelante, los bits CE e IRQ). Las características principales del SPI son:

- El SPI es un estándar de **comunicación en serie** y **síncrono**, lo que quiere decir que paralelamente a la línea de datos (comunicación en serie) existe también una línea de reloj que marca los tiempos de transmisión, cambio de valor y muestreo.
- El SPI es un protocolo de comunicación que trabaja en **modo *full dúplex***, por lo que no sólo existe una línea de datos unidireccional, sino que son dos las líneas unidireccionales de datos: una de entrada y otra de salida.
- El estándar SPI está contemplado para comunicar 2 o más dispositivos, pero siempre existirá uno que sea el de mayor relevancia que el resto; esto es, **Maestro** y **Esclavo** o esclavos. La tarea principal del dispositivo maestro es la de generar la señal de reloj (señal SCLK) a una frecuencia adecuada para todos los dispositivos, y además, si la comunicación se configura para ello, habilitar o deshabilitar cada dispositivo esclavo.
- El SPI, como se puede ahora intuir, necesita cómo mínimo de 3 líneas, aunque normalmente son 4 líneas (4 cables de conexión):
  - el **SCLK** (Serial Clock) o SCK
  - el **MOSI** (Master Output, Slave Input) o SDO
  - el **MISO** (Master Input, Slave Output) o SDI
  - el **NSS** (Slave Select)

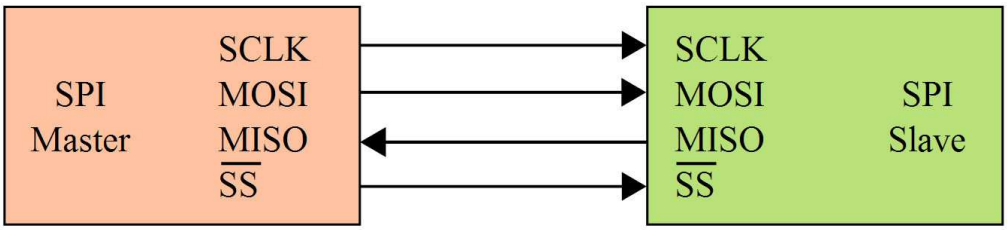


Figura 3-2: Líneas de conexión SPI y su dirección

- La comunicación SPI puede ser comprendida como el envío de información por parte de un emisor y la recepción de ésta por parte de un receptor, o simplemente como un **intercambio de información entre iguales**:
  - Cuando se desee que el Maestro actúe de Emisor y el Esclavo de Receptor, el primero deberá enviar información útil y el segundo recibirla, mientas a la vez envía una información inútil que será desechada (*dummydata*).
  - Cuando se desee que el Dispositivo Maestro actúe de Receptor y el Esclavo de Emisor, el primero deberá enviar información inútil (*dummydata*) y el segundo enviar información útil.
  - En ciertos momentos, como se verá por ejemplo al iniciar cada transferencia de información entre PIC18 y nRF24L01, no existirá emisor y receptor, tan solo transferencia de información útil en ambos casos.
- El proceso de transferencia de información se inicia cuando el Dispositivo Maestro activa la señal NSS (que funciona como señal *Enable*) e inicia la oscilación del reloj.
- La transferencia de datos se realiza **bit a bit**. Cada flanco del reloj impuesto por el Dispositivo Maestro indica el momento de envío-recepción (intercambio) del bit o el momento de cambio de bit de información.
- Las unidades de información sobre las que trabajan los dispositivos son de **1 Byte**. Por eso cada dispositivo dispone de un ***Shift Register*** que por un lado se vacía (empezando por el MSB, que es el primero en enviarse) y por el otro se va llenando (empezando por el nuevo MSB, que es el primero en ser recibido). De este modo se necesitarán además como mínimo ocho ciclos de reloj antes de desactivar la señal NSS y finalizar así la transferencia.

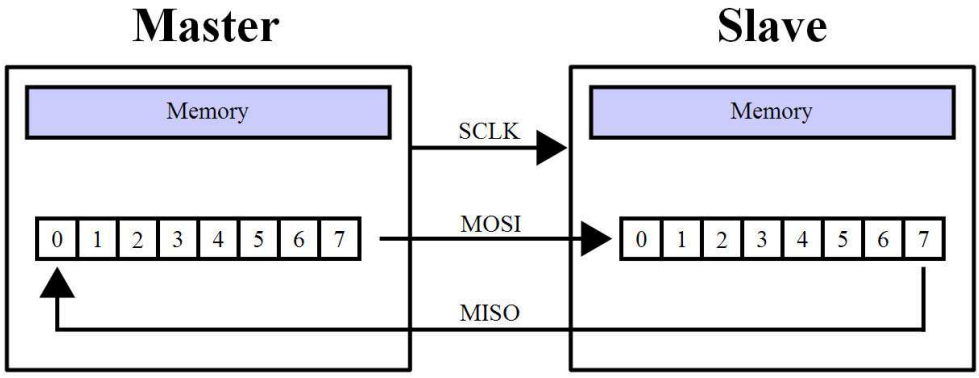


Figura 3-3: Circulación de bits en los *Shift Register*

- Para terminar, cabe destacar que, en función de **2 parámetros de operación** del protocolo SPI, existen **4 modos de trabajo**. Este hecho es de una gran importancia, ya que es necesario que los dos o más dispositivos que se quieran

comunicar trabajen en modos compatibles (si se comunican por ejemplo dos PIC18 es indiferente el modo de trabajo siempre que ambos se configuren igual; pero por el contrario, como se explicará más adelante, cuando se comunican el nRF24L01 y el PIC18, éste último debe adaptarse al único modo de trabajo del primero). Los parámetros de operación son:

- **CPOL** (*Clock Polarity*). Para entornos de *Microchip*: **CKP**=CPOL.
- **CPHA** (*Clock Phase*). Para entornos de *Microchip*: **CKE**=noCPHA.
- Para entornos de *Microchip* existe otro parámetro. Y por tanto así en total **8 modos de trabajo**: **SMP** (*Data sample timing*).

3.2.1. La comunicación SPI en el PIC18F4520

Para llevar a cabo la transmisión o intercambio de bytes mediante SPI, el PIC18F4520 dispone en primer lugar de 2 registros: el **SSPBUF** o *Serial Input Buffer* (Registro de escritura y lectura en el que se carga el dato a enviar y del que se extrae el dato a recibir una vez termina la transferencia) y el **SSPSR** o *Shift Register* (Registro que recoge el byte a transmitir del SSPBUF, lo intercambia vía las líneas SDO y SDI, y da el nuevo dato al SSPBUF activando las **banderas de interrupción** y la **señal de *Buffer Full***):

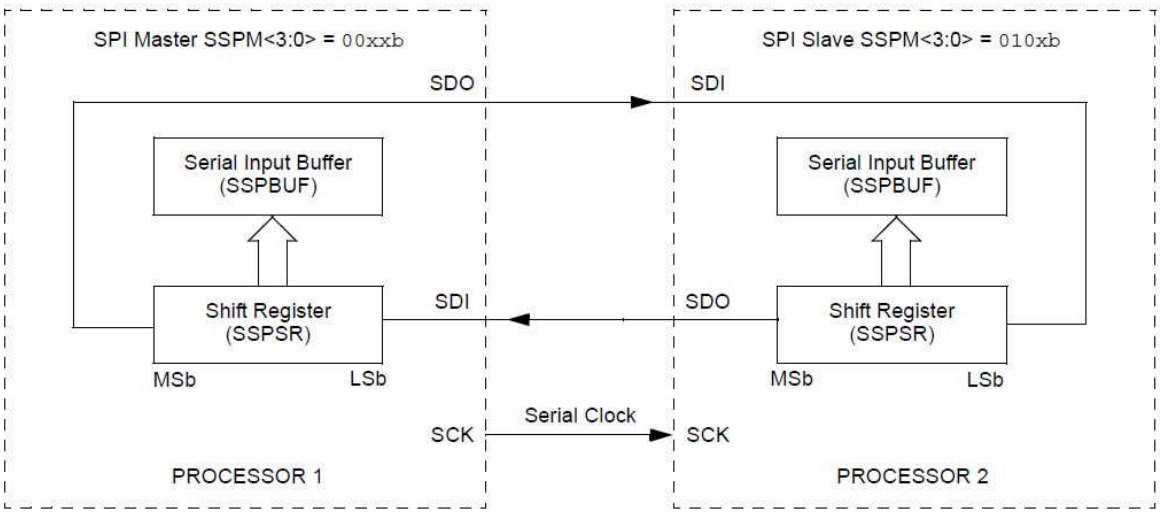


Figura 3-4: Relación entre los registros SSPUF y SSPSR en el PIC18

El PIC18F4520 dispone de diferentes registros de configuración y uso del protocolo SPI dentro del módulo MSSP, así mismo, los pines de cada señal, una vez se habilita la función, son:

- *Serial Data Out* (**SDO**) – RC5/SDO (pin 24)
- *Serial Data In* (**SDI**) – RC4/SDI/SDA (pin 23)
- *Serial Clock* (**SCK**) – RC3/SCK/SCL (pin 18)
- *Slave Select* (**NSS**) – RA5/NSS (pin 7)

Los registros de configuración y seguimiento de la función SPI son básicamente dos, de los cuales sólo se utilizan algunos bits:

- **SSPSTAT** (Bits 7, 6 y 0): *MSSP Status Register*

  bit 7 **SMP**: *Sample bit*
  bit 6 **CKE**: *SPI Clock Select bit*
  bit 0 **BF**: *Buffer Full Status bit (Receive mode only)*

- **SSPCON1** (Bits 7 - 0): *MSSP Control Register*

  bit 7 **WCOL**: *Write Collision Detect bit*
  bit 6 **SSPOV**: *Receive Overflow Indicator bit*
  bit 5 **SSPEN**: *Master Synchronous Serial Port Enable bit*
  bit 4 **CKP**: *Clock Polarity Select bit*
  bit 3-0 **SSPM**<3:0>: *Master Synchronous Serial Port Mode Select bits*
  0101 = *SPI Slave mode*, *clock* = *SCK pin*; N*SS pin control disabled*
  0100 = *SPI Slave mode*, *clock* = *SCK pin*; N*SS pin control enabled*
  0011 = *SPI Master mode*, *clock* = TMR2 output/2
  0010 = *SPI Master mode*, *clock* = FOSC/64
  0001 = *SPI Master mode*, *clock* = FOSC/16
  0000 = *SPI Master mode*, *clock* = FOSC/4

Además de estos bits de configuración, se asocian también al SPI ciertos bits de configuración de las **interrupciones asociadas al MSSP**:

- **SSPIF** (Bit 3) del registro **PIR1**: *MSSP Interrupt Flag bit*.
- **SSPIE** (Bit 3) del registro **PIE1**: *MSSP Interrupt Enable bit*.
- **SSPIP** (Bit 3) del registro **IPR1**: *MSSP Interrupt Priority bit*.

Es importante destacar que se prescindirá del uso de interrupciones relativas al SPI o al nRF24L01 en la elaboración las librerías ni se idearan funciones que requieran de éstas, ya que así quedan libres para que los estudiantes las utilicen para la finalidad que deseen sin ningún tipo de restricción. De modo que el bit **SSPIE** permanecerá siempre a 0.

El bit **SSPIF** indica la misma información que el bit BF del registro SSPSTAT: cuándo el registro SSPBUF tiene disponible un nuevo byte para ser leído (el SSPIF debe ser puesto a cero cada vez que se lee el SSPBUFF, mientras que el bit BF lo hace automáticamente).

Por último se ilustran ahora, según recoge el *Datasheet* del PIC18F4520 (**ANEXO A**), cómo varían las señales SCK, SDO, SDI en función de los modos de trabajo junto con las señales de control SSPIF y NSS:

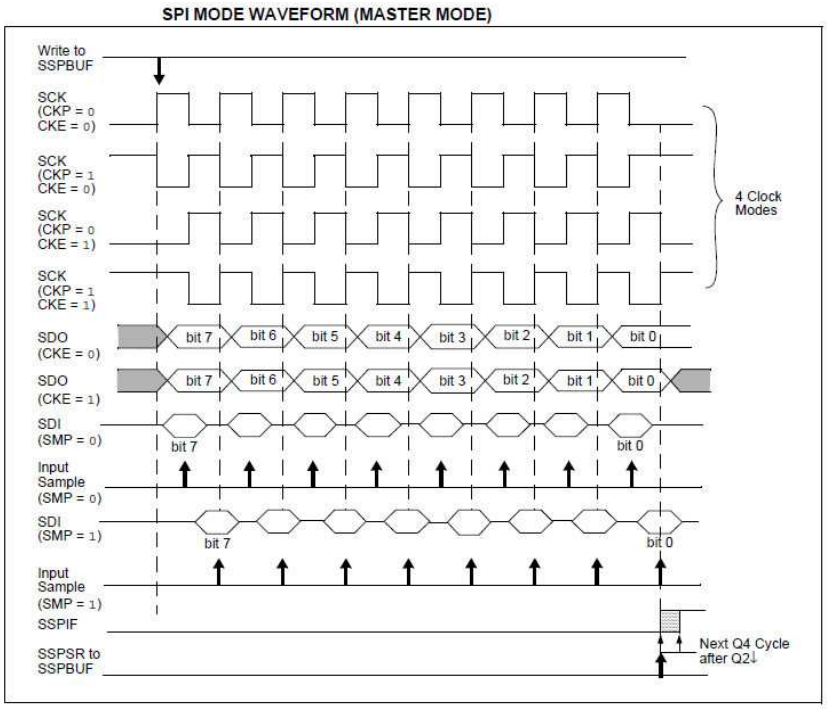


Figura 3-5: Señales SPI en los diferentes modos de trabajo (Dispositivo Maestro)

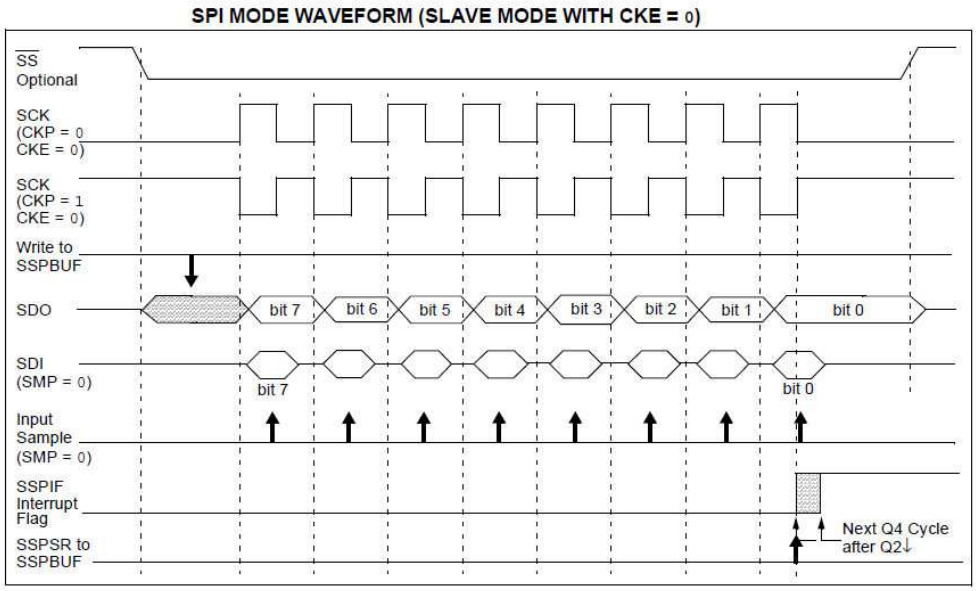


Figura 3-6: Señales SPI para CKE=0 (Dispositivo Esclavo)

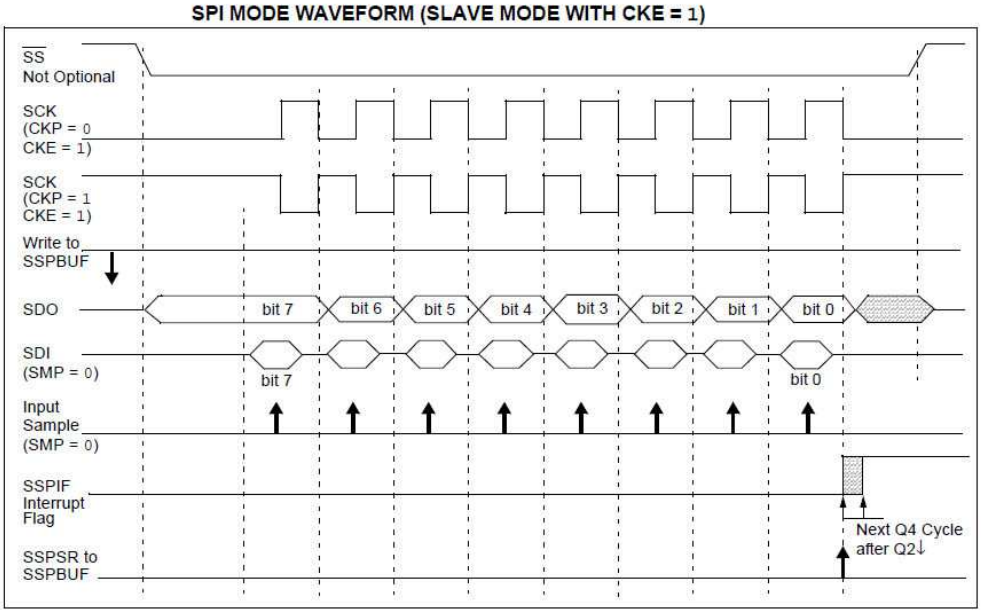


Figura 3-7: Señales SPI para CKE=1 (Dispositivo Esclavo)

Es importante observar qué conjunto de señales corresponde a cada modo parametrizado por las variables CKP, CKE y SMP, ya que para el nRF24L01 no estará documentado el valor de estas variables sino la forma de las señales SPI. Será entonces el PIC18 el que tenga que ser configurado para que la comunicación sea posible.

3.2.2. Programación del SPI: funciones para la comunicación entre dos PIC18

Para el uso de la comunicación SPI, el compilador C18 de MPLAB cuenta con una serie de librerías propias (*spi.h*) que permiten establecer un modo de trabajo y escribir y leer en el registro SSPBUF (tarea que, como ahora se muestra, puede ser resumida en una sola función). A pesar de esto, para la correcta comprensión y configuración del SPI se han elaborado unas funciones propias más intuitivas que además incluyen de forma clara la declaración de la dirección de ciertos bits de puertos. Dichas funciones y algunos códigos de ejemplo para su uso quedan recogidos en el **ANEXO B.** Por consiguiente sólo se describen dos aspectos importantes.

El primero es la definición de la dirección de los puertos en función de si se está habilitando el SPI en modo maestro o esclavo:

Dispositivo Maestro

```
TRISCbits.TRISC3=0;    // SCK out
TRISCbits.TRISC5=0;    // SDO out
TRISCbits.TRISC4=1;    // SDI in
TRISAbits.TRISA5=0;    // CS out
```

Dispositivo Esclavo

```
TRISCbits.TRISC3=1;    // SCK in
TRISCbits.TRISC5=0;    // SDO out
TRISCbits.TRISC4=1;    // SDI in
TRISAbits.TRISA5=1;    // CS in
```

Respecto a la función para el envío y recepción de bytes de datos, ésta se ha definido como *spi_transfer*, un simple intercambio de información, y tiene un parámetro de entrada (el byte a enviar, que se carga en el SSPBUF) y un parámetro de salida (el byte a recibir, que se lee del SSPBUF sólo cuando ha finalizado el intercambio de bytes: BF=1 o SSPIF=1):

```
unsigned char spi_transfer (unsigned char x)
{
        SSPBUF = x;
        while (SSPSTATbits.BF==0);
        return (SSPBUF);
}
```

A lo largo de este proyecto, cuando el **PIC18 (Dispositivo Maestro)** quiere comunicarse con el **nRF24L01 (Dispositivo Esclavo)**, esta función se utiliza de tres modos diferentes:

- Intercambio de información útil en ambos casos:

```
dato_recepcion=spi_transfer(dato_envio);
```

- El PIC envía información al nRF24L01:

```
unsigned char dummydata;
dummydata=spi_transfer(dato_envio);
```

- El PIC recibe información del nRF24L01:

```
unsigned char dummydata;
dummydata=0x00;
dato_recepcion=spi_transfer(dummydata);
```

En el segundo caso el *dummydata* es una información que se desecha al ser recibida; mientras que en el tercero, se trata de un byte cualquiera que se carga en el SSPBUF para hacer posible la transmisión (este paso es imprescindible si se desactiva el control del esclavo por NSS).

Tras la elaboración de estas funciones para dispositivo maestro y esclavo, se han diseñado una serie de programas muy sencillos para comprobar el funcionamiento apropiado de la transmisión de datos y de los PIC18 mediante el control por pulsadores y LEDS de la placa de desarrollo Open18f4520.

Algunas de las verificaciones llevadas a cabo han sido:

- Comprobación de la diferencia de operación al activar o desactivar el **control del esclavo por el bit NSS**: este bit como se ha comentado previamente determina el inicio y final de cada transferencia SPI. Si este control se desactiva, las transmisiones se inician sólo con cargar un nuevo byte en el registro SSPBUF (es recomendable el uso de este bit y necesario para el trabajo con el nRF24L01).
- Trabajo con **diferentes frecuencias de SCK**: entre PIC18 no existe el problema de una frecuencia demasiado alta, ya que soportan con facilidad los MHz de trabajo (no pasará esto, sin embargo, con el nRF24L01, que producirá una tasa de errores elevada a altas frecuencias de trabajo).

- Trabajo con **diferentes modos CKE, CKP y SMP**: sólo con modos iguales la transmisión es efectiva y fiable. Si se configuran modos diferentes la transmisión puede ser imposible o defectuosa.
- Necesidad de interconectar, junto con las señales SCK, SDI, SDO y NSS, las señales de VCC y VDD de ambos PIC18 para que el intercambio de datos sea el esperado y no se vea perturbado por interferencias.

3.3. Código para nRF24L01 de *WaveshareElectronics*

Para la desarrollo de este proyecto en su fase de análisis de antecedentes se ha realizado un búsqueda de códigos en C ya creados anteriormente para el control y uso del dispositivo nRF24L01 de *Nordic*, tras lo cual se han obtenido dos códigos: el primero elaborado por *WaveshareElectronics* y el segundo destinado a un microcontrolador AVR depositado en el portal de código libre *Google Code* (ambos incluidos en el **ANEXO C**).

Las consideraciones fundamentales sobre el primer código, tanto en lo que se ha tomado como referencia como en lo que se ha descartado, son:

- Se trata de un código de ejemplo que no dispone prácticamente de ninguna estructura donde algunas funciones están definidas pero no se utilizan en último término. Además sólo algunos registros y comandos quedan definidos al principio del programa.
- Algunas funciones, tal como están definidas, son poco efectivas y pueden ser simplificadas.
- De todos los parámetros configurables del nRF24L01 ninguno es configurable sin modificar las funciones de la librería.
- No existen funciones que aporten un *feedback* de información sobre el proceso de transferencia o recepción de datos por RF.
- No se respetan las condiciones de *timing* del nRF24L01, o lo que es lo mismo, no se cumplen ciertos retardos o periodos de espera entre algunos cambios de las señales.
- No existen procesos de verificación de la configuración cargada en el nRF24L01 (no se verifica la correcta escritura de los registros), hecho que como se explicará más adelante provoca en muchas ocasiones que el nRF24L01 quede atrapado en el bucle de espera de recepción o envío o actúe de forma inesperada.
- No se previene que el nRF24L01 quede atrapado en este bucle de espera de recepción o envío ni existe un proceso para escapar de este estado.
- No se previene la variabilidad al leer registros del nRF24L01, que a menudo dan resultados falsos.
- No se incluye ningún código extra de verificación del mensaje que permita la detección de errores en la transmisión.

- Por otro lado, algunas líneas de código se han valorado y tomado como ejemplo para la elaboración de las funciones básicas como transmisión/recepción o de los comandos de configuración.

3.4. Código para AVR y nRF24L01 de *Google Code*

Respecto al segundo código hallado (**ANEXO C**), al estar elaborado para un microcontrolador AVR, no se han tenido en cuenta las definiciones de registros propios del microcontrolador. Sin embargo se ha valorado y tomado como referencia la definición inicial de todos los registros y los bits de registros especiales del nRF24L01. Otras consideraciones son:

- La estructura es mucho más intuitiva y las funciones hacen uso de los bits de control de la operación del nRF24L01.
- Se incorporan algunos retardos y condiciones de espera necesarias, pero otros de gran importancia no aparecen.
- De todos los parámetros configurables del nRF24L01 sólo alguno es configurable sin modificar las funciones de la librería.
- No existen funciones que aporten un *feedback* de información sobre el proceso de transferencia o recepción de datos por RF (el código tan solo está contemplado para el caso de buen funcionamiento del nRF24L01 y no contempla las situaciones de error).
- Siguen sin existir procesos para verificar la correcta escritura de los registros del nRF24L01.
- Sigue sin prevenirse el caso de bloque en el bucle de espera de envío o recepción.
- Tampoco se incluye ningún código extra de verificación del mensaje que permita la detección de errores en la transmisión.
- Tampoco se previene la variabilidad al leer registros del nRF24L01 y cualquier lectura se da por correcta cuando a menudo se obtienen lecturas falsas.

4. El módulo *NRF24L01 RF Board*

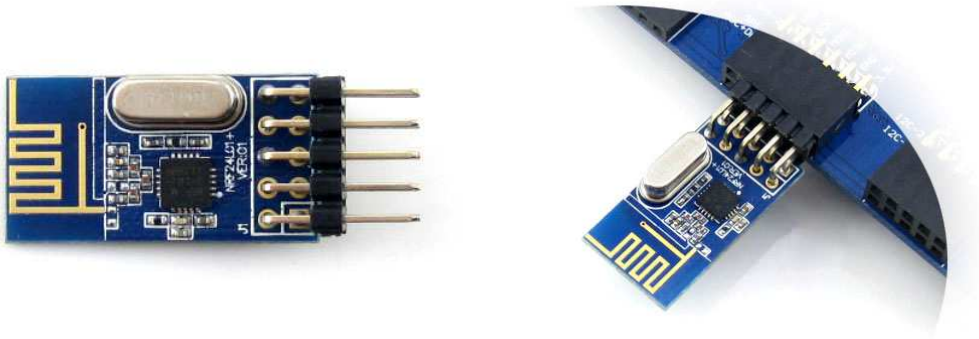


Figura 4-1: Vistas del módulo RF nRF24L01

El nRF24L01 es un chip radio transceptor que está diseñado para trabajar en la banda ISM (*Industrial, Scientific and Medical*) a 2.4 – 2.5 GHz (banda de frecuencias reservada para finalidades no comerciales). El módulo consiste en un sintetizador de frecuencia, un amplificador de potencia, un oscilador de cristal, un modulador, un demodulador y un sistema de *Enhanced ShockBurst*.

La potencia de RF, el canal de frecuencia y el protocolo de configuración son fácilmente programables mediante la interfaz SPI. Por otro lado el consumo del módulo es bajo, oscilando entre los 9mA (a una potencia de salida de -6dBm) y los 12.3 mA (en el modo de trabajo RX que en valor medio es el de mayor consumo). Además es factible un gran ahorro en el consumo gracias al trabajo en los modos de *Power Down*, *Standby-I* y *Standby-II*.

Otras características básicas del transceptor nRF24L01 son:

- Sistema Automático de Acuse de Recibo o *Auto Acknowledgement* sin intervención del microcontrolador gracias al sistema *Enhanced Shockurst*. Como se verá próximamente, esto permite que el dispositivo TX o transmisor reciba automáticamente un comprobante del dispositivo RX o receptor al recibir un paquete de datos.
- Sistema de identificación y verificación del paquete de datos mediante una dirección (de entre 3 y 5 bytes) y un código CRC (si se activa, de 1 o 2 bytes).
- Sistema múltiple de recepción de datos con 6 *Pipes* de llegada.
- Tasa de transmisión de datos por aire (RF) de 1 o 2 Mbps.

- Tasa de transmisión SPI de entre 0 y 10 Mbps.
- 125 canales de RF.
- Chip de 20 pines con un potencial de alimentación requerido de entre 1.9 y 3.6 V.

En el **ANEXO D** se ha reunido toda la información referente al módulo *NRF24L01 RF Board* de *Waveshare* (esquemático y *layout* de la placa) y a su chip (*Datasheet* completo del nRF24L01). De ésta se destacará la definición de pines y las funciones de cada uno en relación a los modos de trabajo y, además, se explicarán brevemente las funciones especiales que caracterizan este dispositivo.

4.1. Pines y modos de trabajo del nRF24L01

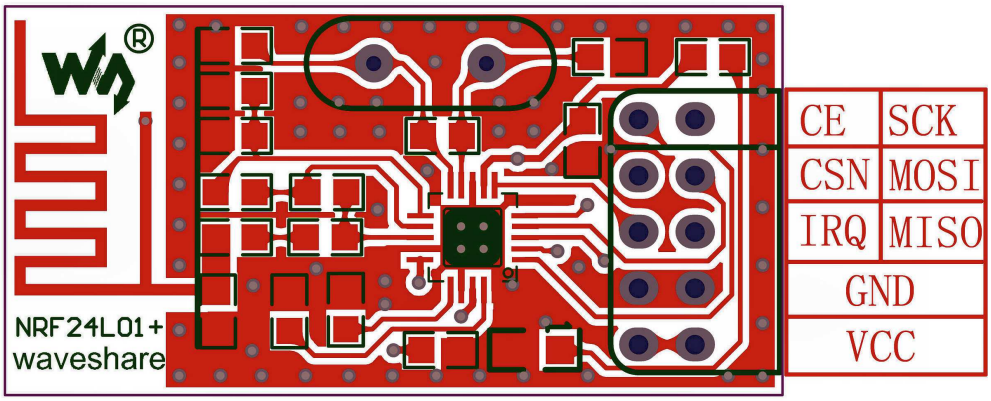


Figura 4-2: *Layout* de la placa nRF24L01 RF *Board*

La comunicación entre nRF24L01 y PIC18 se lleva a cabo mediante seis conexiones, las propias de la comunicación SPI:

- *Serial Data Out* (**SDO o MISO**) – RC5/SDO (pin 24)
- *Serial Data In* (**SDI o MISO**) – RC4/SDI/SDA (pin 23)
- *Serial Clock* (**SCK**) – RC3/SCK/SCL (pin 18)

Y mediante otros 3 bits:

- *Chip Enable* (**CE**) – RA2/AN2/Vref-/CVref (pin 4)
- *Chip Select* (**CSN**) – RA3/AN3/Vref+ (pin 5)
- *Maskable Interrupt Pin* (**IRQ**) – RA4/T0CKI/C1OUT (pin 6)

Pin Name	Direction	TX Mode	RX Mode	Standby Modes	Power Down
CE	Input	High Pulse >10µs	High	Low	-
CSN	Input	SPI Chip Select, active low			
SCK	Input	SPI Clock			
MOSI	Input	SPI Serial Input			
MISO	Tri-state Output	SPI Serial Output			
IRQ	Output	Interrupt, active low			

Figura 4-3: Las señales de comunicación con el nRF24L01 y sus funciones

La señal **CSN** (de salida del PIC) es la misma señal NSS antes definida en el apartado sobre la comunicación SPI. Esta señal marca el inicio y final de cada comunicación entre nRF24L01 y PIC18. Cuando se activa, el nRF24L01 envía su byte de estado mientras que el PIC18 debe enviar una instrucción. Seguidamente PIC18 o nRF24L01 envían N bytes de contenido. Cuando ha finalizado este proceso, la señal CSN vuelve a 1 (se desactiva).

La señal **CE** (de salida del PIC), como se verá ilustrado a continuación, sirve para activar (CE=1) o desactivar (CE=0) los modos RX y TX (uno u otro según esté configurado el nRF24L01). Al activar esta señal, se abandonan los modos *Standby* y, en el caso del modo recepción RX, el nRF24L01 empieza a buscar señales de llegada, o, en el caso del modo transmisor TX, el nRF24L01 empieza a transmitir los datos cargados por el PIC18en la memoria TX FIFO.

La señal **IRQ** (de entrada del PIC) es una señal de interrupción que indica, para el modo RX, cuándo se ha recibido un paquete y puede ser éste recogido por el PIC18; o para el modo TX, cuándo se ha cesado de enviar información al aire (ya sea porque el dispositivo RX lo ha recibido o porque se ha alcanzado un número máximo de retransmisiones permitidas). La lógica de esta señal es negativa: marca la interrupción del proceso con un 0.

Mode	PWR_UP register	PRIM_RX register	CE	FIFO state
RX mode	1	1	1	-
TX mode	1	0	1	Data in TX FIFO
TX mode	1	0	1→0	Stays in TX mode until packet transmission is finished
Standby-II	1	0	1	TX FIFO empty
Standby-I	1	-	0	No ongoing packet transmission
Power Down	0	-	-	-

Figura 4-4: Modos del nRF24L01

El nRF24L01 dispone de **5 modos de trabajo**. El modo ***Power Down*** es el de menor consumo, en el cual la comunicación vía SPI para cargar tanto la configuración como los bytes de datos es posible. En este modo la señal CE no cumple función alguna y es necesario poner a 1 el bit PWR_UP del registro CONFIG para pasar a los modos *Standby*.

Como se verá en próximos apartados, existe un tiempo de espera mínimo una vez se enciende el módulo RF y queda éste en los modos *Standby*. Estos 2 modos se caracterizan por su bajo consumo y por su rápido cambio a los modos RX o TX. Cuando CE se desactiva, el nRF24L01 pasa a descansar en el modo ***Standby-I*** (en el caso de configuración TX, siempre y cuando el dispositivo haya enviado toda la información). El modo ***Standby-II*** se establece (en configuración TX) automáticamente cuando no queda información que enviar y CE sigue activo.

Finalmente al poner CE a 1, el nRF24L01, en función del bit PRIM_RX del registro CONFIG, pasará al **estado de recepción RX** (en el cual buscará sin descanso señales entrantes de la frecuencia y dirección determinadas hasta encontrarlas y activar IRQ) o pasará al **estado de transmisión TX** (en el cual el nRF24L01 enviará el paquete de información una y otra vez hasta recibir el acuse de recibo del dispositivo RX o cesar en el intento: activando en ambos casos la señal IRQ).

Es importante remarcar que durante el diseño de las librerías y sus funciones se ha buscado permanecer **el mayor tiempo posible en los modos de trabajo de menor consumo** para hacer más efectivo el uso del nRF24L01. Esto implica cargar la configuración inicial en el nRF24L01 en el modo *Power Down*, pasar a los modos *Standby* para iniciar la transmisión de forma ágil cuando se solicite, y solamente activar los modos RX y TX cuando sea necesario, esto es, activarlos cuando se quiera iniciar la transmisión o recepción y desactivarlos justo después de recibir la interrupción IRQ. Se contempla además la creación de una función de apagado para volver al modo *Power Down* (lo que no quiere decir volver al estado inicial).

Sobre esta **función de apagado** del nRF24L01 que más a delante se mostrará (desactiva el bit PWR_UP) es necesario comentar que no retorna al estado inicial de funcionamiento del módulo RF, pues los registros mantienen la configuración antes cargada. Sólo se volverá a la configuración por defecto quitando la tensión del nRF24L01, por lo que el uso de las funciones *Hold in Reset* y *Release from Reset* del MPLAB (donde se mantiene la tensión de alimentación) no son recomendables en ciertas fases de la experimentación con el módulo nRF24L01.

4.2. El sistema de *Auto Acknowledgement*

Como se ha comentado, el propio nRF24L01 dispone de un sistema de verificación de transmisión (*acuse de recibo*) gracias a la tecnología *Enhanced ShockBurst* que se ha valorado muy positivamente en este proyecto. A continuación se explica su funcionamiento muy brevemente:

Cuando el **dispositivo TX** inicia el envío de un paquete de datos vía RF, una vez termina la primera transmisión (170us), pasa momentáneamente (otros 170us) a un estado de recepción RX en el que espera un acuse de recibo conforme un dispositivo RX ha recibido esa información. Si el dispositivo TX recibe ese comprobante emite la interrupción IRQ y pasa a modo *Standby*. Si no es así, vuelve a repetir el proceso.

En el caso del **dispositivo RX**, éste monitoriza el aire constantemente hasta encontrar un paquete válido (esto es, de la dirección esperada y con un correcto CRC). En este momento se activa la interrupción IRQ y se pasa momentáneamente (161us) a un estado de transmisión TX en el que envía el acuse de recibo (un identificador del paquete de datos).

4.3. Los *pipes* de recepción y el paquete de datos

La última prestación del módulo nRF24L01 que debe ser comentada para la correcta comprensión de la tarea llevada a cabo en el proyecto es el sistema de recepción de datos vía múltiples canales o *pipes*.

Esta función, como la de verificación por CRC, es posible gracias a la **estructura del paquete de datos** que incluye una dirección de entre 3 y 5 bytes (por defecto 5 bytes):

Preamble 1 byte	Address 3-5 byte	9 bit	Payload 1 - 32 byte	CRC 0/1/2 byte

Figura 4-5: Descripción del paquete de datos

Cada nRF24L01 dispone de **un canal transmisión TX** y de **6 canales de recepción RX**, que le permiten recibir datos de 6 dispositivos diferentes identificando en todo momento su origen gracias a la dirección del paquete. De esta manera, a pesar de que sólo es posible la comunicación por una única dirección a la vez, establecer una red entre dispositivos es sencillo, ya que solamente es necesario reconfigurar el dispositivo transmisor. La siguiente ilustración muestra esta posible configuración entre un dispositivo receptor y seis transmisores, **todos trabajando a la misma frecuencia**:

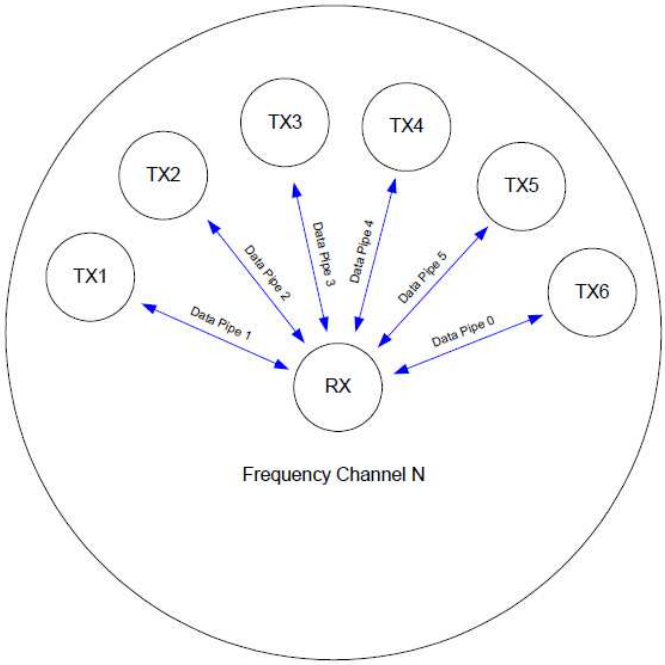


Figura 4-6: Ejemplo de recepción de datos por múltiples *pipes*

5. La librería *UPC_nRF24L01* y sus funciones

Una vez se finalizó el estudio del hardware y la familiarización con el entrono de MPLAB y C18 (elaboración de diversos programas para el uso de periféricos de la placa Open18f4520 y funciones del PIC18), se inició la fase de programación del módulo nRF24L01 para la comunicación por RF.

En los siguientes apartados se explica el proceso de diseño y la finalidad de cada una de las funciones elaboradas, que finalmente se recogieron en una librería llamada ***UPC_nRF24L01***. Para su correcta utilización por parte de los estudiantes además se elaboró **un manual de la librería** (**ANEXO G**).

A continuación se muestra un diagrama indicando el orden de utilización de las funciones principales, es decir, de aquellas que serán utilizadas por los estudiantes para establecer su propia comunicación por RF. Se incluyen también en forma de tabla API:

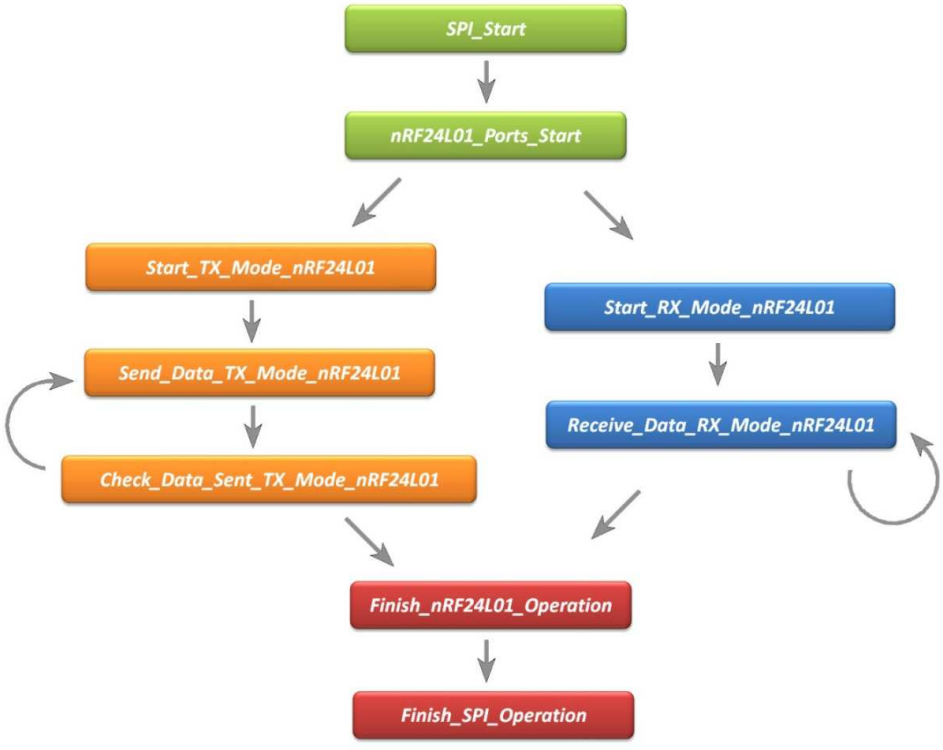


Figura 5-1: Diagrama de uso de las funciones principales

Function Name	Description
`SPI_Start`	This function initializes the SPI hardware setting the PIC18 as the master device.
`nRF24L01_Ports_Start`	This function initializes the extra lines required for communicating with nRF24L01.
`Start_TX_Mode_nRF24L01`	This function initializes and configures the nRF24L01 as a TX device (transmitter).
`Start_RX_Mode_nRF24L01`	This function initializes and configures the nRF24L01 as a RX device (receiver).
`Send_Data_TX_Mode_nRF24L01`	This function loads data to send in the nRF24L01 and starts RF transmission.
`Check_Data_Sent_TX_Mode_nRF24L01`	This function checks RF transmission and returns a report of the process.
`Receive_Data_RX_Mode_nRF24L01`	This function activates the nRF24L01 to search incoming signals and returns a report of the process and the data received.
`Finish_nRF24L01_Operation`	This function turns off the nRF24L01 to reduce consumption.
`Finish_SPI_Operation`	This function deactivates the SPI of the PIC18 MSSP hardware.

Tabla 5-1: Funciones principales de la librería *UPC_nRF24L01*

6. Funciones de comunicación con el nRF24L01

La primera fase de trabajo con el módulo nRF24L01 consistió en establecer y verificar, siguiendo los modelos elaborados anteriormente para la comunicación entre dos PIC18, la correcta comunicación entre nRF24L01 y PIC18 vía SPI. Tras solucionar algunos errores o funcionamientos inesperados se elaboraron las funciones de carga y lectura de los registros internos del nRF24L01, teniendo que mejorarlas en algunos aspectos para hacerlas fiables.

Sobre el trabajo en MPLAB en lenguaje C18 cabe comentar que se utilizaron las librerías propias del PIC18F4520 (*p18f4520.h*), las librerías de retardos (*delays.h*) y el archivo de los bits de configuración del PIC18 (*config.h*) que queda adjuntado en el **ANEXO E** junto con las librerías elaboradas.

6.1. La función *SPI_Start*

```
void SPI_Start (unsigned char Clock_Frequency)
{
        TRISCbits.TRISC3=0;
        TRISCbits.TRISC4=1;
        TRISCbits.TRISC5=0;

        SSPCON1 = (SSPCON1 & 0xF0) | Clock_Frequency;
        SSPCON1bits.CKP=0;
        SSPSTATbits.CKE=1;
        SSPSTATbits.SMP=1;

        PIE1bits.SSPIE=0;
        IPR1bits.SSPIP=0;
        PIR1bits.SSPIF=0;

        SSPCON1bits.SSPEN=1;
}
```

La primera función que se elabora es la de inicialización de la función SPI en el PIC18, llamada ***SPI_Start***. En ella en primer lugar se establece la configuración de puertos SPI de dispositivo maestro: SCK es por tanto una salida, SDI una entrada y SDO una salida.

Seguidamente se determina, en el registro SSPCON1, que el dispositivo será Maestro y la frecuencia de reloj SPI a partir del parámetro de entrada de la función. Se establece el modo de trabajo en los registros SSPCON1 y SSPSTAT a partir de los bits CKP, CKE y SMP; se desactivan las interrupciones del MSSP y finalmente se habilita el SPI.

Se puede ver cómo en esta función, que se trata de la función definitiva, **el modo de operación queda fijado como CKP=0, CKE=1 y SMP=1**. Pero en un principio estos tres parámetros eran, junto con la frecuencia de reloj, parámetros de entrada de la función; y en los primeros usos que se le dio a la función se configuraron los tres a 0. Con esta configuración se observó que a la hora de leer registros del nRF24L01 (mediante funciones que se mostrarán más adelante), la información que se recibía era de otro registro y no del demandado:

- Al solicitar el contenido del registro 0000 0001 se mostraba el del 0000 0000.
- Al solicitar el contenido del registro 0000 0101 se mostraba el del 0000 0010.
- Al solicitar el contenido del registro 0000 0111 se mostraba el del 0000 0011.

Se podía intuir que el último bit no era correctamente enviado, y tras observar detenidamente el diagrama del *SPI Timing* del nRF24L01, en el que se observó que la lectura de cada bit se realizaba con el primer flanco del reloj, se llegó a la conclusión de que se trataba de un problema en la fase del reloj del SPI: CPHA debía ser puesto a 0, esto quiere decir, **CKE=1** (pues CKE=noCPHA) para que así el diagrama de señales del PIC18 fuera acorde con el del nRF24L01.

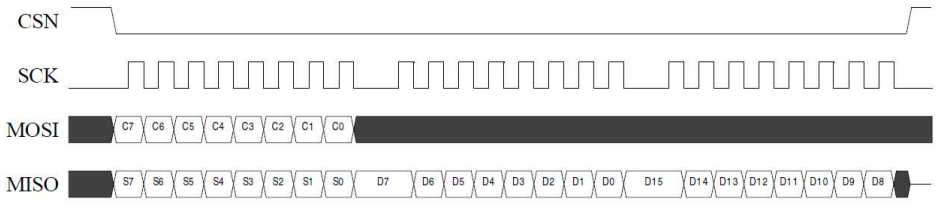


Figura 6-1: Formas de las señales CSN y SCK en la comunicación SPI

Sobre el momento de la toma de muestra de los datos (valor del bit SMP) no se encontró información en el *Datasheet* del nRF24L01. De modo que se llevaron a cabo diversos procesos de lectura/escritura para determinar el mejor modo de trabajo para el SPI: el bit SMP debía ser puesto a 1, además, cualquier otro modo de trabajo del PIC18 (los otros siete modos) era incompatibles con el nRF24L01.

Por último, el único parámetro que queda configurable es la **frecuencia de SPI**, que puede ser de 1MHz, **250KHz** o **62.5KHz**. Durante el proceso de diseño de las funciones objeto de este proyecto se trabajó en todo momento a la frecuencia de 62.5KHz para evitar errores. Posteriormente se elevó ésta a 250KHz sin ninguna consecuencia o repercusión en el normal funcionamiento del sistema. Por último, al elevarse la frecuencia a 1MHz, la tasa de errores de funcionamiento tanto en dispositivo trasmisor como receptor aumentó considerablemente; a pesar de que según el fabricante el nRF24L01 soporta hasta 10Mbps por SPI.

Se desaconseja por tanto, para un uso fiable y robusto del módulo RF, el uso de la frecuencia de 1MHz. Por otro lado, dado que el uso de estas comunicaciones por RF se dará con finalidad docente y no entre dispositivos comerciales, esto tampoco supone un problema, pues no se necesitarán altas velocidades de transmisión.

6.2. Funciones *nRF24L01_Ports_Start* y *SPI_Transfer*

```
void nRF24L01_Ports_Start (void)
{
        ADCON1=0x0F;
        TRISAbits.TRISA2=0;
        TRISAbits.TRISA3=0;
        TRISAbits.TRISA4=1;

        CSN=1;
        CE=0;
}

unsigned char SPI_Transfer(unsigned char byte_to_send)
{
        SSPBUF = byte_to_send;
        while(PIR1bits.SSPIF==0);
        PIR1bits.SSPIF=0;

        return(SSPBUF);
}
```

La función ***nRF24L01_Ports_Start*** no es más que la inicialización de las señales CE (Salida), CSN (Salida) y IRQ (Entrada) y su definición en el estado inicial: ambas se deshabilitan recordando que CSN mantiene lógica negativa. Además, para garantizar el buen funcionamiento de los puertos, se establecen todos ellos como digitales y no analógicos mediante la configuración del registro ADCON1.

Por otro lado, se crea la misma función ***SPI_Transfer*** para el intercambio de bytes entre un dispositivo y otro; sólo que en este caso, por ningún motivo en particular, se ha establecido como señal de llegada de nuevo byte la *Interrupt Flag* en lugar del bit BF. Excepto por la puesta a cero del SSPIF, el proceso es el mismo.

6.3. Funciones de lectura y escritura de registros simples del nRF24L01

Una vez listas las funciones de inicialización y habilitación de puertos y transferencia SPI ya es posible la comunicación con el nRF24L01 mediante el sistema de instrucciones establecido por el fabricante:

- Con la puesta a 0 de la señal CSN se inicia una comunicación SPI.
- El PIC18 envía una **instrucción (1 byte de longitud)** mientras el nRF24L01 envía por su parte el contenido de su registro de estado (**STATUS**, de 1 byte).
- Seguidamente, en función de la instrucción del PIC18, se finaliza la operación (poniendo a 1 la señal CSN) o se transmiten **N bytes** de datos en una u otra dirección. Si, por ejemplo, el PIC18 ha demandado al nRF24L01 el valor de su registro EN_AA, éste, después de enviar el valor de su registro STATUS enviará seguidamente el del EN_AA.
- Así una comunicación implicará **N+1 ciclos** completos de SPI. Cuando esta comunicación finalice la señal CSN deberá ponerse de nuevo a 1.

Los diagramas siguientes muestran el flujo de datos para el proceso de lectura de un registro (esquema superior) y el proceso de escritura en el nRF24L01 (esquema inferior), siendo la línea MOSI la salida de datos del PIC18 y la línea MISO la entrada de datos. El byte ***Ci*** es la instrucción de lectura del PIC18, mientras que el ***Si*** es el STATUS del nRF24L01 y el ***Di*** son los bytes de datos del contenido del registro solicitado. Se producen en este caso tres ciclos SPI completos.

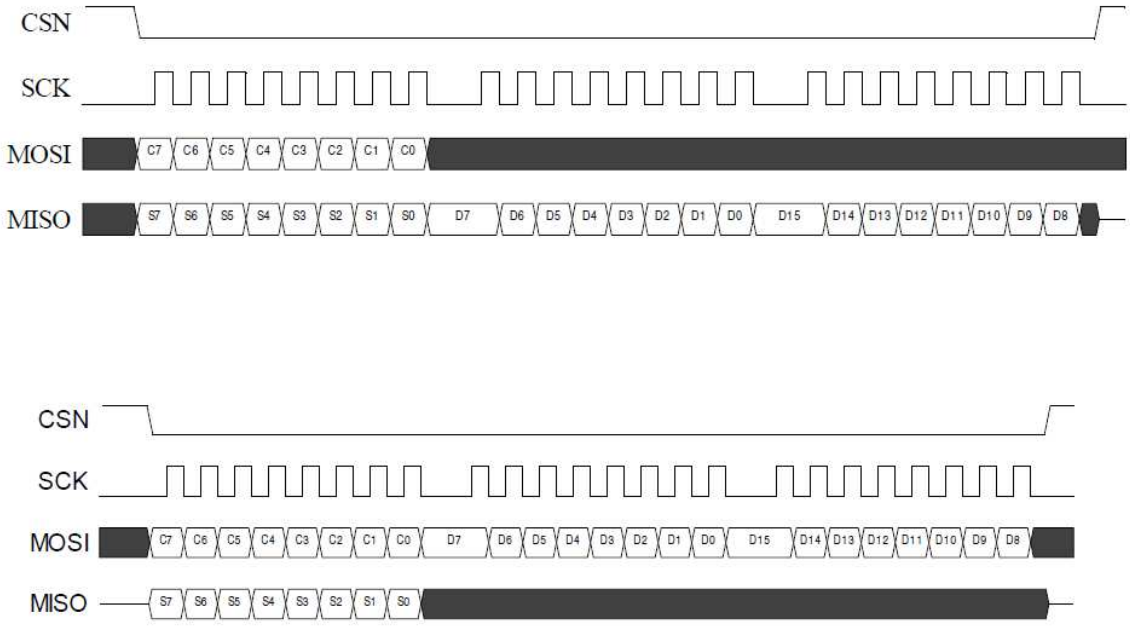


Figura 6-2: Procesos de lectura y escritura por SPI entre PIC18 y nRF24L01

En la siguiente tabla se muestran las 9 instrucciones de que dispone el PIC18 para comunicarse con el nRF24L01. Para que le entendimiento de la tarea de este proyecto sea progresivo, se irán comentando estas instrucciones a medida que se requiera su uso.

Instruction Name	Instruction Format [binary]	# Data Bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read registers. AAAAA = 5 bit Memory Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write registers. AAAAA = 5 bit Memory Map Address *Executable in power down or standby modes only.*
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation will always start at byte 0. Payload will be deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Used in TX mode. Write TX-payload: 1 – 32 bytes. A write operation will always start at byte 0.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, i.e. acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last sent payload. Packets will be repeatedly resent as long as CE is high. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

Figura 6-3: Instrucciones del nRF24L01

Las primeras dos instrucciones que serán utilizadas, **R_REGISTER** y **W_REGISTER**, tienen como finalidad respectivamente leer y escribir registros de **1 byte** del nRF24L01. Dado que los registros vienen indicados desde la dirección 0x00 a la dirección 0x17 (esto es, 23 registros en total), son sólo necesarios 5 bits para nombrarlos. Por eso, ambas instrucciones de 1 byte, consisten en 3 bits seguidos de los 5 bits de la dirección del registro.

Para entender mejor lo que en adelante se explica es recomendable acudir al **ANEXO D** para observar en el *Datasheet* del nRF24L01 el ***Memory Map***. En él se muestra cada registro con su dirección de 5 bits y con la función de cada uno de sus bits. En las librerías elaboradas en este proyecto, para facilitar la comprensión del estudiante que las utilice, **se ha definido además cada instrucción y cada registro en el archivo *.h* y los bits de aquéllos más importantes**, como serán el registro CONFIG y el registro STATUS.

De los 23 registros existentes 21 son **simples**, esto es, de 1 byte a ancho; mientras que 2 son **especiales** (el 0x0A y el 0x0B) al tener 3, 4 o 5 bytes de ancho según la configuración del registro 0x03. Este hecho supone en la práctica un tratamiento diferente para ambos grupos, ya que los registros especiales requerirán más ciclos SPI y sus funciones en C18 requerirán el uso de memoria dinámica.

Por este motivo se ha optado por **establecer dos grupos de funciones de lectura y escritura**: las de los registros simples y las de los registros especiales (registros de direcciones RF). Se mostrarán y comentarán ahora las funciones del primer grupo:

```
unsigned char Read_nRF24L01_Register (unsigned char Register_Address)
{
        unsigned char Register_Content, nRF24L01_Status;

        CSN=0;
        nRF24L01_Status=SPI_Transfer(R_REGISTER+Register_Address);
        Register_Content=SPI_Transfer(0x00);
        CSN=1;

        return(Register_Content);
}
```

La función ***Read_nRF24L01_Register*** es la encargada de obtener el contenido de un registro determinado del nRF24L01 (de 1 byte). Ha sido diseñada con un parámetro de entrada (la dirección correspondiente al registro que se quiere leer) y con un parámetro de salida o de retorno (el contenido de ese registro). Ambos parámetros se definen como *unsigned char*, esto es, un byte.

Como se ha explicado anteriormente, el proceso de instrucción se inicia con la activación de la señal CSN. Posteriormente se realiza un intercambio SPI en el que se recibe el *nRF24L01_Status* (contenido del registro 0x07 del nRF24L01) y se envía la instrucción:

R_REGISTER + *Register_Address* = 0b00000000 + *Register_Address*

Por ejemplo, para leer el registro 0x12: 0b00010010

Acto seguido, como se ha demandado el contenido de un registro de 1 byte, se inicia otra transferencia SPI en la que el PIC18 recibe esa información útil mientras envía una información inútil o *dummydata* que será ignorada por el nRF24L01. Ese byte útil se asigna a la variable *Register_Content*, que será devuelta por la función después de finalizar el proceso SPI desactivando la señal CSN.

Cabe comentar que se descarta aquí, como en muchas otras funciones que se verán más adelante, la información del registro STATUS del nRF24L01, pues su información (referente a los resultados de transmisión o recepción por RF) no es relevante en estos procesos.

```
void Write_nRF24L01_Register (        unsigned char Register_Address,
                                      unsigned char Register_Content        )
{
        unsigned char dummydata, nRF24L01_Status;

        CSN=0;
        nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);
        dummydata=SPI_Transfer(Register_Content);
        CSN=1;

        while(Read_nRF24L01_Register(Register_Address)!=Register_Content)
        {
                CSN=0;
                nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);
                dummydata=SPI_Transfer(Register_Content);
                CSN=1;
        }
}
```

La función ***Write_nRF24L01_Register*** es la encargada de escribir un contenido en un registro del nRF24L01. Tiene dos parámetros de entrada: la dirección del registro y el contenido a cargar respectivamente, ambos de 1 byte (*unsigned char*). El principio de la función es equivalente al de la función de lectura, sólo que en este caso la instrucción es:

W_REGISTER + *Register_Address* = 0b00100000 + *Register_Address*

Por ejemplo, para escribir el registro 0x12: 0b00110010

Posteriormente se realiza un **proceso de verificación**. En éste se lee el registro en cuestión y se compara el contenido leído con el contenido previamente escrito. En el caso de que la verificación falle, se repite el proceso de escritura una y otra vez hasta que se dé por buena la verificación.

La incorporación de este proceso de verificación es una de las soluciones importantes propuestas en este proyecto de programación ante ciertos errores de gran importancia en el funcionamiento del nRF24L01. Concretamente en este caso, **a menudo los registros no se escribían correctamente**, lo que provocaba un funcionamiento inesperado.

Lo más común era que el dispositivo (tanto RX como TX) **quedase bloqueado en el proceso de espera de la interrupción IRQ** (*while(IRQ);*), de modo que se introdujo en este bucle un proceso de verificación de los registros (se mostraba registro a registro por los LEDs de la placa entre periodos de espera). Se observó que a menudo algunos no cumplían la configuración cargada previamente, especialmente el registro CONFIG, que tenía dado de baja el bit PWR_UP.

Ésta era la causa del bloqueo en la espera de la interrupción IRQ una vez se había habilitado CE para entrar en el modo TX o el modo RX, pues al cargarse mal este bit PWR_UP, el nRF24L01 permanecía en el modo *Power Down*, y por mucho que se habilitase CE no pasaba a los modos TX y RX en los cuales se podía dar la interrupción IRQ de llegada o envío de datos.

Finalmente, tras implementar este proceso de verificación en la escritura y volver a ejecutar la visualización de los registros por los LEDs, no se volvió a detectar ningún error de este tipo y se dio la solución propuesta por válida (se considera, por otro lado, que supone un sobrecoste de tiempo de ejecución importante en el programa, pero que debe ser asumido para hacerlo efectivo y fiable).

6.4. Funciones lectura y escritura del registro STATUS

Dentro del grupo de los registros simples, de 1 byte de contenido, el registro STATUS (con dirección 0x07) requiere un trato especial tanto a la hora de ser leído como modificado. Este registro y sus bits quedan definidos en archivo *.h* de las librerías elaboradas y en la tabla siguiente:

Registro 0x07: **STATUS**	
Bit 7	*Reserved*
Bit 6	**RX_DR**
Bit 5	**TX_DS**
Bit 4	**MAX_RT**
Bits 3 – 1	**RX_P_NO**
Bit 0	**TX_FULL**

Tabla 6-1: Bits del registro STATUS

En configuración de **dispositivo RX** (receptor), el bit **RX_DR** marca la llegada de un nuevo paquete de datos que puede ser al momento obtenido vía SPI. Además, hasta que no se dé de baja este bit por el programa, los 3 bits del registro **RX_P_NO** indican la procedencia de esta información recibida, esto es, el *pipe* por el que han llegado (así que el RX_P_NO puede valer desde 0b000 hasta 0b101 indicando el *pipe* de la información disponible o 0b111 en el caso de no haber recibido paquete alguno o de este haber sido ya leído y dado de baja el bit RX_DR.

En configuración de **dispositivo TX** (transmisor), el bit **TX_DS** marca el correcto envío de un paquete de datos a un dispositivo RX, esto es, como el sistema de *Auto Acknowledgement* permanecerá siempre activado, que se ha recibido el acuse de recibo del receptor. Por otro lado, si este comprobante de correcto envío no se recibe después de haberlo intentado un máximo de veces (las programadas en los bits ARC del registro SETUP_RETR), se activa el bit **MAX_RT**. Además se incorpora el bit **TX_FULL** para notificar cuándo se ha llenado la memoria TX FIFO y alertar en el caso que el programa siga cargando en ella información.

La interrupción **IRQ** antes mencionada en tantas ocasiones responde a los primeros bits del registro STATUS (MAX_RT, TX_DS y RX_DR), siempre que uno de ellos se active también

lo hará la interrupción. En el caso del dispositivo transmisor se tendrá además que analizar el registro STATUS para encontrar la causa de la interrupción IRQ, ya que puede ser originada tanto por el envío exitoso (TX_DS) como por el fallido (MAX_RT).

Los bits RX_P_NO y TX_FULL son bits de sólo lectura, por lo que no pueden ser modificados por el programa, mientras que los bits MAX_RT, TX_DS y RX_DR sólo pueden modificarse para ser puestos a 0. Este proceso de puesta a 0 de uno de estos bits se ejecuta escribiendo un 1, y es imprescindible realizarlo una vez se recoge su información para permitir futuras comunicaciones RF tanto en el dispositivo receptor como transmisor (dar de baja IRQ).

Se muestran finalmente las funcione de lectura y escritura del registro STATUS:

```
unsigned char Read_nRF24L01_Status (void)
{
        unsigned char nRF24L01_Status;

        CSN=0;
        nRF24L01_Status=SPI_Transfer(NOP);
        CSN=1;
        return(nRF24L01_Status);
}
```

Para obtener el contenido del registro STATUS sólo es necesaria la instrucción nula NOP antes mostrada en la tabla de funciones. Esta es una instrucción que no manda orden alguna al nRF24L01, pues sólo sirve para que en el intercambio se reciba su estado. De modo que la función ***Read_nRF24L01_Status*** sólo consiste de un ciclo SPI en el que se obtiene y retorna (a diferencia de las funciones vistas hasta ahora) la variable nRF24L01_Status.

Es importante destacar que en el proceso de comunicación RF conocer el contenido del registro STATUS es sólo necesario en ciertos momentos puntuales. Por este motivo se ha optado en el proyecto por descartarlo y no retornarlo en la mayoría de funciones: sólo de esta función se obtiene su contenido. De este modo se simplifica la sintaxis de las funciones y se hace más intuitivo todo el programa.

```
void Write_nRF24L01_Status (unsigned char Register_Content)
{
        unsigned char dummydata, nRF24L01_Status;
        CSN=0;
        nRF24L01_Status=SPI_Transfer(W_REGISTER+STATUS);
        dummydata=SPI_Transfer(Register_Content);
        CSN=1;
}
```

La función ***Write_nRF24L01_Status*** de escritura o modificación del registro STATUS es equivalente a la función *Write_nRF24L01_Register* sólo que en este caso ya queda definido el parámetro *Register_Address* como STATUS sin necesidad del parámetro de entrada. Otra diferencia es que en este caso no se debe introducir un contenido que deba ser escrito sino tan solo el bit que se quiera eliminar, y es por eso que no se hace uso del proceso de verificación, pues en todo momento el contenido de STATUS es diferente al bit que se desea eliminar. Un ejemplo de uso de la función sería:

Si se desea eliminar el bit RX_DR:

```
Write_nRF24L01_Status (RX_DR);
```

o equivalentemente:

```
Write_nRF24L01_Status (0b01000000);
```

6.5. Funciones de lectura y escritura de registros especiales del nRF24L01

Las funciones que ahora se muestran son las dedicadas a los dos registros especiales o registros de direcciones que tiene el nRF24L01 (0x0A y 0x0B). Estos corresponden a las direcciones de los *pipes* RX 0 y 1; y son registros que pueden tener entre 3 y 5 bytes de ancho en función de la configuración cargada en el registro SETUP_AW.

```
void Read_nRF24L01_Address_Register (        unsigned char TX_RX_Address_Width,
        unsigned char Register_Address,      unsigned char *Register_Content )
{
        unsigned char i, nRF24L01_Status;

        CSN=0;
        nRF24L01_Status=SPI_Transfer(R_REGISTER+Register_Address);
        for(i=0; i<(TX_RX_Address_Width+0b10); i++)
        {
                Register_Content[i]=SPI_Transfer(0x00);
        }
        CSN=1;
}
```

```
void Write_nRF24L01_Address_Register (       unsigned char TX_RX_Address_Width,
    unsigned char Register_Address,          unsigned char *Register_Content       )
{
    unsigned char dummydata, i, nRF24L01_Status;
    unsigned char Process_Finished;
    unsigned char Address_Verification[5];

    Process_Finished=0;
    while(!Process_Finished)
    {
       CSN=0;
       nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);
       for(i=0; i<(TX_RX_Address_Width+0b10); i++)
       {
               dummydata=SPI_Transfer(Register_Content[i]);
       }
       CSN=1;

       Read_nRF24L01_Address_Register(TX_RX_Address_Width, Register_Address,
       Address_Verification);
       Process_Finished=1;
       for(i=0; i<(TX_RX_Address_Width+0b10); i++)
       {
               if(Register_Content[i]!=Address_Verification[i])
               {
                       Process_Finished=0;
               }
       }
    }
}
```

Dado que en este caso el contenido de cada registro no es un simple byte, se ha estimado más conveniente tratar esta información mediante tablas de bytes y punteros. El proceso sin embargo es equivalente al de las funciones para registros simples, excepto en:

- No se retornan funciones, sino que las tablas de bytes son datos de entrada y salida definidos por punteros.
- El proceso requiere de más ciclos SPI, en total 1+N siendo N el ancho en bytes de las direcciones.
- Otro parámetro de entrada es justamente esta longitud de las direcciones, codificado según: 0b01 para 3 bytes, 0b10 para 4 bytes, y 0b11 para 5 bytes.
- El proceso de verificación requiere ahora comparar no sólo 1 byte sino hasta 5 bytes. De modo que no se decide si es correcta la escritura con una mera comparación: es necesario definir una variable de validación del proceso que tenga en cuenta hasta 5 comparaciones**.**

6.6. Lectura del *RX Payload* y escritura del *TX Payload*

Las siguientes funciones que se han desarrollado en el proyecto son las referentes a la **carga de datos a transmitir vía RF** (o *payload*). Como se puede ver en la tabla de instrucciones, la lectura y escritura de estos *payload* tiene sus propias instrucciones: la instrucción de lectura se ejecutará solamente en la configuración de dispositivo RX, y la de escritura en la de dispositivo TX. El ancho de estos *payload* será de entre 1 y 32 bytes, determinándolo en los registros del 0x11 al 0x16 y también en las mismas funciones de escritura o lectura para determinar el número de ciclos SPI necesarios en el proceso.

Es importante aquí destacar que las funciones que se muestran a continuación no son las que originalmente se utilizaron en las primeras fases de la experimentación con el módulo RF, pues se incluye ahora en ellas **un sistema de verificación de envío de datos *checksum***.

Tras realizar algunas pruebas sin sistema de verificación alguna, se observó que en muchas ocasiones se confirmaba el envío y la recepción del paquete de datos pero al visualizar el mensaje éste era erróneo. Una vez implementado el *checksum* se volvieron a realizar estas pruebas y muy pocas veces se produjo un error en los bytes enviados no alertado por la verificación *checksum*. Se dio entonces por válido el sistema de verificación.

Para hacer más versátil la comunicación RF de cara a la docencia y para analizar el efecto de un simple *checksum* de 1 byte en ella, se ha optado por posibilitar su habilitación o deshabilitación; aunque se recomienda encarecidamente su uso. Por otro lado, el

estudiante siempre podrá elaborar su propio código redundante para aumentar la distancia de *Hamming* y así hacer más fiable la comunicación mediante la detección de errores.

La primera función en cuestión se ha llamado ***Read_nRF24L01_RX_Payload***, y dada su extensión tan solo se comentarán sus líneas más importantes:

- Se crea una variable de entrada *Enable_Checksum* que determina si el paquete de datos que se recibe incorporara un byte de *checksum* al final.
- En el caso de *Enable_Checksum*=0, el PIC18 simplemente lee la información en tantos ciclos SPI como ancho sea el *RX payload* (determinado en la variable de entrada *TX_RX_Payload_Width*). La función devuelve un 1 conforme la información leída se da por buena.
- En el caso de *Enable_Checksum*=1, el proceso es algo más complejo, pues cada byte que se recibe, no sólo se memoriza sino que se suma al anterior para obtener el *checksum del dispositivo RX*. Finalmente el último byte recibido se memoriza como el *checksum del dispositivo TX*.
- De la comparación entre el *TX_Checksum* y el *RX_Checksum* se obtiene la verificación de la comunicación RF y es retornada en la variable *Checksum_Conclusion*.

La segunda función se ha llamado ***Write_nRF24L01_TX_Payload***, y sobre ella debe considerarse:

- Se crea del mismo modo la variable de entrada *Enable_Checksum* que determina si el paquete de datos que se enviará debe incorporar un byte de *checksum* al final.
- En el caso de *Enable_Checksum*=0, el PIC18 simplemente transmite la información al nRF24L01 en tantos ciclos SPI como ancho sea el *TX payload* (determinado en la variable de entrada *TX_RX_Payload_Width*).
- En el caso de *Enable_Checksum*=1, cada byte que se transmite vía SPI se incrementa en el *TX_Checksum* para obtener el *checksum* del *payload*.
- Finalmente se realiza una transferencia SPI más en la que se envía este byte de verificación que será luego analizado por el dispositivo RX.

Una consideración final que debe ser tomada en cuenta al utilizar el *checksum byte* es que la máxima información útil del *payload* se reduce de 32 bytes a 31, por lo que el parámetro de entrada *TX_RX_Payload_Width* deberá ser como máximo 31 en el caso de *Enable_Checksum*=1. En el manual de librerías elaborado (**ANEXO G**) se especifica detalladamente esta condición.

7. Funciones de configuración del nRF24L01

En la siguiente fase de diseño de las librerías se implementaron las dos funciones de inicialización de los dispositivos RX y TX: ***Start_RX_Mode_nRF24L01*** y ***Start_TX_Mode_nRF24L01***. Se buscó en todo momento *aprovechar toda la versatilidad que el nRF24L01 ofrece* haciendo configurables todos sus parámetros de trabajo de forma sencilla e intuitiva. De esta manera se podrá experimentar con el módulo RF y se podrá así mismo comprobar sus limitaciones fácilmente.

Ambas funciones en primer lugar ejecutan otra función encargada de garantizar que los bits MAX_RT, TX_DS y RX_DR estén a 0 y que las memorias FIFO RX y FIFO TX queden vacías. Esto es sólo necesario en el caso de un mal uso del nRF24L01 (concretamente un error de uso de la señal CE), pero se buscan **funciones que garanticen una inicialización correcta y completa** sea cual sea la configuración o las operaciones previamente ejecutadas. Se muestra esta función a continuación:

```
void Reset_nRF24L01_Status_and_nRF24L01_Payloads (void)
{
        unsigned char nRF24L01_Status;

        CSN=0;
        nRF24L01_Status=SPI_Transfer(FLUSH_TX);
       CSN=1;
        Delay1TCY();
        CSN=0;
        nRF24L01_Status=SPI_Transfer(FLUSH_RX);
        CSN=1;

        Write_nRF24L01_Status(RX_DR);
        Write_nRF24L01_Status(TX_DS);
        Write_nRF24L01_Status(MAX_RT);
}
```

La función ***Reset_nRF24L01_Status_and_nRF24L01_Payloads*** emite al nRF24L01, en primer lugar, las dos instrucciones de borrado de las memorias FIFO RX y TX (ambas separadas por la desactivación y la activación de la señal CSN y por un retardo de 1µs para garantizar el tiempo mínimo de CSN inactivo determinado en el *Datasheet* del nRF24L01). Posteriormente se borran los 3 bits del registro STATUS por el procedimiento explicado en el apartado 6.4.

Para inicializar el dispositivo receptor se debe ejecutar, después de las funciones de inicialización del SPI y de inicialización de los puertos para el nRF24L01, la función ***Start_RX_Mode_nRF24L01***. Ésta no se muestra a continuación por su extensión pero se comentan algunos de sus puntos principales:

- En primer lugar se desactiva la señal CE para **garantizar que el nRF24L01 permanezca en los modos de bajo consumo** (*Power Down* antes del encendido y *Standby* tras la configuración del registro CONFIG).
- Tras ejecutar la función *Reset_nRF24L01_Status_and_nRF24L01_Payloads*, se configuran uno a uno todos los registros del nRF24L01 *implicados en el modo de trabajo RX* (se deja de configurar por ejemplo el registro SETUP_RETR, que sólo es utilizado en el modo TX).
- La mayoría de registros se configuran a partir de parámetros de entrada de la función determinados por el usuario.
- Hay registros que se inicializan mediante un solo parámetro de entrada y otros mediante dos o hasta tres parámetros que son colocados de forma apropiada.
- Sólo 2 registros (EN_AA y EN_RXADDR) se cargan de forma automática y no modificable. Se ha decidido implementarlo así porque:
  - Se desea permitir el uso de todos los *pipes* por el usuario.
  - Se desea forzar, para todos los *pipes*, el sistema de *Auto Acknowledgement* dada su gran utilidad antes comentada.
- Se puede observar además la diferencia de trato de las dos funciones de lectura y escritura de registros especiales que aparecen:
  - Requieren del parámetro de ancho de dirección (*TX_RX_Address_Width*)
  - No reciben un parámetro sino un puntero (uso de memoria dinámica).
- En las últimas líneas se configura el registro CONFIG del siguiente modo:
  - Se habilita el código CRC de forma forzada dada su utilidad y conveniencia a la hora de detectar errores en las comunicaciones RF.
  - Se deja a la decisión del usuario si este código CRC será de 1 ó 2 bytes.
  - Se pone a 1 el bit de PWR_UP
  - Se determina el modo de dispositivo: modo RX.
- Finalmente se aplica **la primera de las condiciones de *timing*** o de procesos de espera especificadas por el fabricante del nRF24L01 que en ciertos códigos antecedentes no son respetadas: se fuerza un periodo de espera de 1.5ms, tiempo necesario entre el paso del modo *Power Down* a los modos *Standby*.

Para inicializar el dispositivo transmisor se debe ejecutar la función llamada ***Start_TX_Mode_nRF24L01***. Su estructura es similar a la anterior, excepto en los puntos que se explican a continuación:

- Solamente se ponen a 1 en este caso los registros relacionados con la operación de transmisión TX, por lo que se descartan gran cantidad de registros (especialmente los relacionados con los *pipes* RX).

- La mayoría de registros se configuran a partir de parámetros de entrada de la función determinados por el usuario.
- De igual manera se cargan de forma automática y no modificable los registros EN_AA y EN_RXADDR. Se ha implementado así en el dispositivo transmisor porque éste, si se desea hacer uso de la prestación de *Auto Acknowledgement*, debe tener habilitada la recepción de información de cara a recibir los acuses de recibo.
- No se carga dirección alguna en esta función de inicialización, pues se ha **considerado oportuno reservar esta acción para la función de envío de un paquete de datos**. De este modo si se quiere enviar datos a 2 o más dispositivos (establecer una red de comunicación) es mucho más sencillo implementarlo y no se necesita cargar en el dispositivo TX toda la configuración inicial.
- En la puesta en marcha del nRF24L01 se desactiva el bit PRIM_RX del registro CONFIG para determinar en este caso el modo TX.

En próximos apartados y en el manual de la librería elaborada (**ANEXO G**) se explica el correcto uso de estas funciones. Así mismo, se recomienda consultar las funciones anteriores en el **ANEXO E** para conocer qué registros concretamente intervienen en las configuraciones TX y RX.

8. Funciones de envío y recepción de datos

Tras comprobar el buen funcionamiento de las dos operaciones anteriores de inicialización del dispositivo, el siguiente paso en el proyecto se encaminó a implementar las órdenes de envío y recepción. Se consideró oportuno elaborarlas inicialmente conforme a dos requerimientos:

- Que optimizasen el consumo permaneciendo el mayor tiempo posible en los modos *Standby*.
- Que aportasen la mayor información posible sobre el proceso en curso y sobre el resultado de la operación de envío/recepción.

En estas funciones que aquí se tratan se dedicó **la mayor parte del esfuerzo de diseño del proyecto**. En primer lugar porque, a pesar de que el proceso de envío y recepción es simplemente el definido por el fabricante, se tuvo que buscar el mejor sistema para devolver la información del proceso (tanto en RX como en TX) de modo que fuera intuitivo el uso de ésta.

En segundo lugar, fue en estas funciones donde se tuvo que poner remedio a algunos de los errores ocasionales de funcionamiento del nRF24L01 más críticos. El método de deducción y de posterior resolución de estos comportamientos anómalos se comenta seguidamente a cada función en la que se le pone remedio, pero a modo de síntesis, bajo el punto de vista del autor, **los fallos críticos del nRF24L01 son tres**:

- **El primer mal funcionamiento** del nRF24L01 ya ha sido comentado y resuelto en pasados apartados, que consistía en que **el nRF24L01 a menudo falla en la escritura de registros** (problema solucionado mediante a la implantación de un proceso de verificación).
- **El segundo problema** consiste en que, una vez se entra en los modos RX y TX y se espera la activación de la interrupción IRQ (ya será por recepción de paquete o por envío fallido o exitoso), **esta interrupción IRQ nunca llega** (no se activan los bits MAX_RT, TX_DS o RX_DR) a pesar de la correcta configuración de todos los registros del nRF24L01: el nRF24L01 queda simplemente bloqueado.
- **El tercer fallo** se produce en la **lectura de ciertos registros**, que en la mayoría de las veces se obtiene el contenido correcto pero en otras **se obtiene un contenido distorsionado**. Este hecho se pudo probar creando un bucle infinito en el que se leía un registro y constantemente se cargaba en los LEDs de la placa: se observaba que a veces se mostraba un contenido imposible, por lo que se achacó a problemas de ruido en el nRF24L01 seguramente dado un mal

diseño del dispositivo (esta prueba se realizó apagando la RF y apartando la placa de posibles perturbaciones y el problema persistía).

La primera función se ha llamado ***Receive_Data_RX_Mode_nRF24L01***, y es la encargada de, en el modo RX, activar la recepción de datos, cargar el RX *payload* obtenido en una tabla de bytes determinada por el usuario y notificar el resultado de la operación mediante un parámetro de salida. Este resultado podrá ser:

- Error en la espera de la interrupción IRQ.
- Recepción de paquete de datos satisfactoria (*checksum* correcto).
- Recepción de paquete de datos insatisfactoria (*checksum* incorrecto).

Se comentan a continuación las líneas más importantes de la función:

- La función se inicia son la activación de la señal CE para que el nRF24L01 pase del modo *Standby* al modo RX y empieza a buscar señales entrantes de RF. Este paso de un modo a otro requiere de cómo mínimo una espera de 130µs (**segunda condición de *timing***), mientras que la señal CE debe mantenerse activa como mínimo 10µs (**tercera condición de *timing***). Este segundo retardo no es necesario dado que ya se cumple sobradamente en el anterior, pero se ha decidido implementarlo para que futuros estudiantes tengan noción de esta condición en posibles modificaciones futuras del código.

```
CE=1;
Delay10TCYx(13);
Delay10TCYx(1);
```

- El siguiente paso en la función es el proceso de espera de la interrupción IRQ, que en los primeros diseños de la función se implementó simplemente como una línea: `while(IRQ);` como se ha comentado previamente. Ahora sin embargo, dentro de este bucle, se ha incorporado u**n contador que permite que la señal IRQ se analice cada 30µs** y que, después de un **periodo de espera máximo determinado por el usuario**, en tal caso el bucle se rompa y se comunique el error. Este tiempo máximo, expresado en décimas de segundo, viene determinado por el parámetro de entrada *Max_Waiting_ds*, que puede valer entre 0 y 255.

  Se soluciona así de forma satisfactoria para el caso del dispositivo RX el segundo problema del nRF24L01, pues se evita que el programa quede bloqueado. El periodo de refresco a la hora de analizar la señal IRQ es muy bajo, de modo que no se pierde efectividad; y al poder ajustar el estudiante el tiempo máximo de espera,

le permite crear un margen cómodo de trabajo para manejar el dispositivo RX y saber al momento cuando falla la IRQ del nRF24L01.

Es importante además destacar que otra solución a este fallo hubiese podido ser activar el *Watchdog* del PIC18 para el dispositivo RX, pero se considera que tiene un tiempo máximo muy corto (entorno a los cuatro segundos). De esta manera, además de disponer de un amplio abanico de tiempos, se deja el *Watchdog* disponible para otros usos.

```
i=0; j=0;
while(IRQ)
{
        i++;
        if(i==3333)
        {
                i=0; j++;
        }
        if(j==Max_Waiting_ds)
        {
                IRQ_Error=1;
                break;
        }
}
```

- Una vez termina el proceso de espera, ya exista error del nRF24L01 o recepción válida o inválida, la señal CE se desactiva para volver al modo *Standby* y se respeta el tiempo mínimo de 4µs entre un flanco de CE y la activación de CSN (**cuarta condición de *timing***).

  Posteriormente, se lee en el registro STATUS el *pipe* de origen de los bits RX_P_NO, se extrae el *payload* recibido del nRF24L01 y se almacena el resultado de la verificación *checksum* en la variable *Checksum_Conclusion*. Para dar de baja la interrupción IRQ se elimina el 1 en el bit RX_DR del registro STATUS y se retorna el **informe de la operación** resumido en un byte:

  Bit 4: IRQ correcta o Error en la IRQ
  Bits 1 – 3: *Pipe* de llegada de la información leída
  Bit 0: *Checksum* correcto o incorrecto

```
        CE=0;
        Delay10TCYx(1);

        nRF24L01_Status=Read_nRF24L01_Status();
        Origin_Pipe=(nRF24L01_Status & RX_P_NO);
        Checksum_Conclusion=Read_nRF24L01_RX_Payload(Enable_Checksum,TX
                _RX_Payload_Width, RX_Payload);
        Write_nRF24L01_Status(RX_DR);
        return (IRQ_Error*0b10000+Origin_Pipe+Checksum_Conclusion);
```

En lo que al envío de datos en el modo TX respecta, se han creado dos funciones: una sencilla para el envío de un cierto *payload* con una cierta dirección, llamada ***Send_Data_TX_Mode_nRF24L01***, y otra para la obtención de la información del proceso, llamada **Check_Data_Sent_TX_Mode_nRF24L01**.

```
void Send_Data_TX_Mode_nRF24L01(            unsigned char Enable_Checksum,
     unsigned char TX_RX_Address_Width,     unsigned char *TX_Address,
     unsigned char TX_RX_Payload_Width,     unsigned char *TX_Payload     )
{
     Write_nRF24L01_Address_Register (TX_RX_Address_Width, TX_ADDR,
        TX_Address);
     Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P0,
        TX_Address);
     Write_nRF24L01_TX_Payload (Enable_Checksum, TX_RX_Payload_Width,
        TX_Payload);

     CE=1;
     Delay10TCYx(13);
     Delay10TCYx(1);
     CE=0;
     Delay10TCYx(1);
}
```

La primera se inicia con la definición de la dirección de transmisión, que es cargada en dos registros: en el TX_ADDR y en el RX_ADDR_P0 (ya que para el dispositivo transmisor el *pipe 0* es el canal de recepción del acuse de recibo o *Auto Acknowledgement*; y ambos deben contener la misma dirección).

Posteriormente se escribe en la memoria FIFO del nRF24L01 el *payload* a transmitir y se activa la señal CE para pasar al modo TX. Después del tiempo mínimo de transición de modos y de CE activo (condiciones segunda y tercera de *timing*) se desactiva la señal CE. Como se muestra en la figura 4-4, el nRF24L01 permanecerá en el modo TX hasta que el envío se realice o falle.

Por último, la función ***Check_Data_Sent_TX_Mode_nRF24L01*** consta de dos procesos: el proceso de espera de la interrupción IRQ, y el proceso de su eliminación y determinación del resultado de la operación.

Igual que en la función *Receive_Data_RX_Mode_nRF24L01*, se ha implementado un contador que determine un tiempo máximo en espera de la interrupción IRQ, sólo que en este caso no viene determinado por el usuario sino por una condición que se deduce y calcula de la configuración del nRF24L01: si se establece un caso extremo (en el registro SETUP_RETR) de un número máximo de retransmisiones de 15 (ARC) y un retardo máximo entre retransmisiones de 4ms; **después de $(15+1)\cdot 4 = 64$ ms no hay motivo por el que la IRQ no salte**, bien por haber transmitido satisfactoriamente el paquete (TX_DS) o por haber alcanzado el máximo de retransmisiones (MAX_RT).

Sea cual sea la configuración del transmisor, si se sobrepasa este tiempo de espera, se puede determinar con certeza que se ha producido un fallo en el nRF24L01. En tal caso, el segundo proceso mencionado no se ejecutará y directamente se retornará el informe de error.

En el caso de que el proceso se desarrolle de forma correcta y salte la interrupción IRQ, se inicia la segunda parte de la función:

```
while(!IRQ)
{
        nRF24L01_Status=Read_nRF24L01_Status();
        if(nRF24L01_Status & MAX_RT)
        {
                TX_Operation_Result=0b00;
                TX_Retransmit_Counter=Read_nRF24L01_Register(OBSERVE_TX) & 0x0F;

                Write_nRF24L01_Status(MAX_RT);
        }
        else if(nRF24L01_Status & TX_DS)
        {
                TX_Operation_Result=0b01;
                TX_Retransmit_Counter=Read_nRF24L01_Register(OBSERVE_TX) & 0x0F;

                Write_nRF24L01_Status(TX_DS);

        }
}
```

En ella **se pone remedio al tercer error crítico del nRF24L01** que provoca que a menudo la lectura de un registro sea incorrecta, en este caso, la lectura del registro STATUS mediante la función *Read_nRF24L01_Status*.

En **un primer diseño** de la función, el contenido del bucle `while(!IRQ)` se encontraba suelto en la función (a continuación del bucle `while(IRQ)` y con el mismo orden). De modo que, una vez se había dado la interrupción IRQ, se leía el registro STATUS para saber si provenía del bit MAX_RT o del bit TX_DS (mediante las expresiones `nRF24L01_Status & MAX_RT` y `nRF24L01_Status & TX_DS`). Seguidamente en cada caso se definía el resultado de la operación, se registraba el número de retransmisiones llevadas a cabo (últimos 4 bits del registro OBSERVE_TX) y se daba de baja el bit del registro STATUS causante de la interrupción.

En las pruebas que se llevaron a cabo con este primer diseño se observó que en muchas ocasiones el **transmisor TX notificaba un fallo** en el envío (MAX_RT) cuando en realidad el dispositivo **receptor RX había recibido el *payload***. Se intuyó que el problema se daba en la línea `if(nRF24L01_Status & MAX_RT)`, pues se accedía a este *if* a pesar de que el bit MAX_RT no debía estar activado.

Para analizar mejor la situación, se decidió comprobar si la lectura de este registro funcionaba correctamente mostrándolo por los LEDs de la placa y actualizándolo continuamente. Cuando los LEDs debían permanecer con el valor 0b00001110, el valor *on reset* del registro STATUS, a veces parpadeaban mostrando a menudo valores como 0b11111110 o 0b00000000 (siendo el primero incluso imposible ya que el primer bit sólo permite un 0).

Esta experimentación se realizó sin ninguna configuración previa cargada en el nRF24L01 (antena RF apagada y todos los registros configurados con el valor de *power on reset*), intentando mantener la placa aislada de posibles interferencias y con una comunicación SPI a baja frecuencia; de modo que se considera este mal funcionamiento **un error en el funcionamiento del nRF24L01**.

Se realizó la misma experimentación con otros registros y efectivamente en ocasiones su lectura fallaba y daba contenidos equivocados; por lo que se concluyó que ésta era la causa del fallo de la función que aquí se trata. Por este motivo se decidió incluir esas líneas de código en un `while(!IRQ)`: este proceso de lectura y comparación del registro STATUS se repite una y otra vez mientras IRQ siga activado (esto es, IRQ=0), y sólo se podrá salir de él cuando anteriormente se haya eliminado el bit causante de la interrupción y, en consecuencia, se hay definido correctamente el resultado de la operación.

Dicho de otra manera: si por casualidad la variable *nRF24L01_Status* adopta un valor falso y se accede al primer *if* (se afirma que la operación ha fallado y se borra el bit MAX_RT) cuando en realidad el proceso ha sido exitoso, como al borrar el bit MAX_RT la interrupción IRQ aún seguirá activa, el proceso se volverá a repetir. Hasta que no se elimine el bit correcto y se determine correctamente el resultado de la operación no se finalizará la función.

Finalmente, este resultado de la operación se resume en 1 byte:

Bits 2 - 5:	Número de retransmisiones llevadas a cabo
Bits 0 - 1:	0b00 ➔ Fallo en el envío del *payload*
	0b01 ➔ *Payload* enviado correctamente
	0b11 ➔ Error IRQ

9. Funciones de finalización

Las últimas dos funciones diseñadas en el proyecto son las encargadas del apagado del nRF24L01 manteniendo la configuración cargada (puesta a 0 del registro PWR_UP mediante la expresión `previous_config_register & 0b11111101`) y de la deshabilitación de la función SPI en el PIC18 (dando de baja el bit SSPEN del registro SSPCON1):

```
void Finish_nRF24L01_Operation (void)
{
        unsigned char previous_config_register;

        previous_config_register=Read_nRF24L01_Register(CONFIG);
        write_nRF24L01_Register(CONFIG, previous_config_register & 0b11111101);
}

void Finish_SPI_Operation (void)
{
        SSPCON1bits.SSPEN=0;
}
```

10. Uso de la librería *UPC_nRF24L01*: test y validación del sistema de comunicaciones

Tras dar por válidas las funciones elaboradas, que no sólo aprovechan eficientemente las prestaciones del módulo RF nRF24L01 sino que incluso previenen algunos de sus comportamientos anómalos y aportan al usuario un amplio control sobre el proceso de comunicación, éstas se han agrupado y conformado en una librería que se ha llamado ***UPC_nRF24L01*** (**ANEXO E**), compuesta por su archivo cabecera (*UPC_nRF24L01.h*) y su archivo de código (*UPC_nRF24L01.c*). Ambos archivos, y en general los proyectos de MPLAB elaborados en este trabajo, se incorporan en el CD adjunto.

Por otro lado, se ha elaborado un **manual de la librería** (**ANEXO G**) destinado al buen uso de todas sus funciones, tanto por su orden de ejecución como por la especificación de todos sus parámetros de entrada y salida. A continuación se tratará un ejemplo de uso de estas librerías (código de test y validación: **ANEXO F**), concretamente el que ha servido para la prueba y validación del trabajo realizado, tanto de las comunicaciones RF como de las funciones concretas elaboradas. Como se comprobará, este código ejemplo implanta algunos procesos útiles para visualización de las diferentes fases de la comunicación por RF, y hace uso de todos los datos de *feedback* que aportan las funciones.

10.1. Código de test y validación: *Dispositivo TX*

Como se ha indicado anteriormente, este código ejemplo de test y validación del proceso se ha recogido en el **ANEXO F** (código del dispositivo transmisor y del receptor). A continuación se comentarán, para el dispositivo TX, cada uno de sus bloques por separado prestando atención a su utilidad y al modo en que han sido configurados.

```
#include<p18f4520.h>
#include<delays.h>
#include"config.h"
#include"UPC_nRF24L01.h"

void main (void)
{
        unsigned char direccion_transmision[5] = {0xB2, 0xB2, 0xB3, 0xB4, 0x01};
        unsigned char mensaje[8] = {  0b00000001, 0b00000010, 0b00000100, 0b00001000,
                                      0b00010000, 0b00100000, 0b01000000, 0b10000000};
        unsigned char informe;

        TRISB=0x00;
        TRISD=0xFF;
```

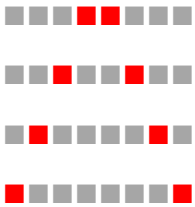
En primer lugar, se incluyen en el código la librería del PIC18F4520, la librería estándar de retardos, el archivo de parámetros de configuración del PIC18 y finalmente la librería elaborada en el proyecto.

En segundo lugar, dentro de la función principal del **dispositivo transmisor**, se define **una dirección de envío** del paquete de datos vía RF mediante una tabla de 5 bytes (el ancho de las direcciones será de 5 bytes) y **un mensaje** sencillo fácil de visualizar mediante los LEDs de la placa (el ancho del *payload* de envío será de 8 bytes). Así mismo se define una variable de 1 byte en la que se cargará la información del proceso de envío.

Como se hará uso de los LEDs del puerto B y de un pulsador del puerto D, se define el primero como puerto de salida y el primero como puerto de entrada.

```
        while(1)
        {
                LATB=0b00011000;
                Delay10KTCYx(30);
                LATB=0b00100100;
                Delay10KTCYx(30);
                LATB=0b01000010;
                Delay10KTCYx(30);
                LATB=0b10000001;
                Delay10KTCYx(30);
```

Seguidamente se establece una **secuencia visual de LEDs** que indica al usuario el inicio o reinicio del proceso (indicación visual que será especialmente útil cuando, como se comentará más adelante, el proceso se repita una y otra vez para detectar algún posible error o mal funcionamiento). Esta secuencia es:



```
SPI_Start(0b10);
nRF24L01_Ports_Start();
Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10, 1, 8);
Send_Data_TX_Mode_nRF24L01(1, 0b11, direccion_transmision, 8, mensaje);
informe=Check_Data_Sent_TX_Mode_nRF24L01();
Finish_nRF24L01_Operation();
```

A continuación se produce todo el **proceso de inicialización y comunicación**: inicialización del SPI y de los puertos para el nRF24L01, inicialización del dispositivo como TX, envío del paquete de datos, verificación del envío y apagado del nF24L01. En la variable *informe* se almacena el resultado del proceso que después será mostrado al usuario. La configuración cargada en el dispositivo transmisor, tanto en la función *Start* como en la función *Send*, es:

- El reloj SPI se configura a baja frecuencia para evitar posibles errores, esto es, a 62.5KHz.
- El ancho de dirección (5 bytes) se indica mediante *0b11*.
- La frecuencia utilizada corresponde a la definida por *0b1000000*.
- El *Data Rate* se configura a 1Mbps (introduciendo un 0) para evitar posibles fallos.
- La potencia de antena se eleva al máximo (0dBm) para garantizar el intercambio de datos en esta fase de prueba mediante el valor *0b11*.
- Se activa la ganancia LNA y se determinan 2 bytes de CRC.
- Se elimina el retardo entre retransmisiones (0b0000) y se marca como 10 el máximo número de retransmisiones permitidas.
- Finalmente se habilita el proceso de verificación por *checksum* y se determina el ancho de *payload* a 8 bytes.
- En la función de envío *Send_Data_TX_Mode_nRF24L01* se reafirma la habilitación del *checksum* y el ancho de banda de dirección RF y *payload*.

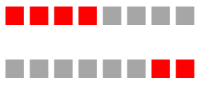
Además se indica la tabla de bytes de la que el nRF24L01 obtendrá la dirección de envío y el *payload* a transmitir.

```
                if((informe & 0b11)==0b01)
                {
                        LATB=0b11110000;
                       Delay10KTCYx(100);
                        LATB=informe>>2;
                }
                else if((informe & 0b11)==0b00)
                {
                        LATB=0b00001111;
                       Delay10KTCYx(100);
                        LATB=informe>>2;
                }
                else if((informe & 0b11)==0b11)
                {
                        LATB=0b00111100;
                }

                while(PORTDbits.RD1);
        }
}
```

Por último se obtiene la información del proceso y se muestra al usuario mediante la visualización en los LEDs. El código finaliza con la condición de que el programa sólo se reinicie en el caso de que el usuario accione el pulsador S0. Los posibles resultados y su código de visualización son:

- **Correcto envío del paquete de datos**: se encienden los primeros cuatro LEDs durante un segundo y posteriormente se muestra el número de retransmisiones que han sido necesarias (por ejemplo, tres).



- **Fallo en el envío**: se encienden los últimos cuatro LEDs durante un segundo y posteriormente se muestran el número de retransmisiones llevadas a cabo (que para este caso, deben ser 10).

- **Error de interrupción IRQ**: se encienden los cuatro LEDs del medio (no han existido retransmisiones del paquete).

Sobre el código del dispositivo transmisor sólo cabe destacar que, mediante la determinación de la dirección de envío por RF en la función *Send*, por un lado se necesita repetir algunos parámetros de configuración, pero por otro se agiliza el proceso en el caso de que **un mismo dispositivo quiera transmitir un paquete a dos o más receptores**; pues tan solo se debe copiar la misma línea de código variando la dirección sin tener que volver a repetir el proceso de inicialización del dispositivo.

10.2. Código de test y validación: *Dispositivo RX*

```
#include<p18f4520.h>
#include<delays.h>
#include"config.h"
#include"UPC_nRF24L01.h"

void main (void)
{
        unsigned char direccion_recepcion_A[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
        unsigned char direccion_recepcion_B[5]={0x11, 0x11, 0x11, 0x11, 0x11};
        unsigned char mensaje_llegada[8];
        unsigned char informe;

        TRISB=0x00;
        TRISD=0xFF;
```

El código elaborado para el **dispositivo receptor** RX empieza con la misma inclusión de los archivos cabecera. En este caso se definen cuatro variables: dos direcciones RF (de 5 bytes de ancho), una tabla sin contenido en la que se volcará el contenido del *payload* recibido (de 8 bytes de ancho) y finalmente un byte que recogerá el resultado de la operación del dispositivo receptor. Por último se define la dirección de los puertos B y D para el uso de los LEDs y del pulsador.

```
        while(1)
        {
                LATB=0b00011000;
                Delay10KTCYx(30);
                LATB=0b00100100;
                Delay10KTCYx(30);
                LATB=0b01000010;
                Delay10KTCYx(30);
                LATB=0b10000001;
                Delay10KTCYx(30);

                SPI_Start(0b10);
                nRF24L01_Ports_Start();
                Start_RX_Mode_nRF24L01(       0b11, 0b1000000, 0, 0b11, 1, 1,
                       direccion_recepcion_A, direccion_recepcion_B, 0x00, 0x00, 0x00,
                       0x00, 1, 8, 8, 0, 0, 0, 0      );
                informe=Receive_Data_RX_Mode_nRF24L01(1, 100, 8, mensaje_llegada);

                Finish_nRF24L01_Operation();
```

Tras la misma **secuencia visual de inicio** del programa, se inicializa la función SPI, los puertos para el nRF24L01 y el dispositivo como receptor. Seguidamente se demanda la búsqueda de paquetes entrantes y se carga la información del proceso en la variable *informe*. Para finalizar la operación del nRF24L01 se ejecuta la función de apagado.

La configuración que se carga en las funciones de inicialización y recepción es:

- El reloj SPI se configura a la misma baja frecuencia (podría ser diferente a la del dispositivo transmisor).
- El ancho de dirección (5 bytes) se indica mediante *0b11*.
- La frecuencia utilizada corresponde a la definida por *0b1000000* (necesariamente la misma que la del dispositivo transmisor),
- El *Data Rate* se ajusta a 1Mbps (0 como parámetro de entrada).
- La potencia de antena se eleva al máximo (0dBm) para garantizar el intercambio de datos en esta fase de prueba mediante el valor *0b11*.
- Se activa la ganancia LNA y se determinan 2 bytes de CRC.
- Se determina como dirección del *pipe 0 RX* la dirección antes definida como *dirección_recepción_A* (que es la misma que la de transmisión del dispositivo TX, de modo que en este código ejemplo en canal de entrada de datos para el dispositivo RX será el *pipe* 0).
- Se determina como dirección del *pipe 1 RX* la dirección antes definida como *dirección_recepción_B*.
- El resto de direcciones (*pipes* del 2 al 5) se determinan mediante un solo byte, pues su dirección completa consiste en los MSBytes de la dirección del *pipe* 1 y un LSByte particular. En este caso se ponen estos LSByte a 0x00.

- Se habilita el proceso de verificación por *Checksum* (necesariamente igual que en el transmisor).
- Se determina cuánto espacio se reserva en la memoria RX FIFO para cada *pipe* de llegada. En este caso, se reservan 8 bytes para el *pipe* 0, 8 bytes para el *pipe* 1 y 0 bytes para el resto de *pipes*.

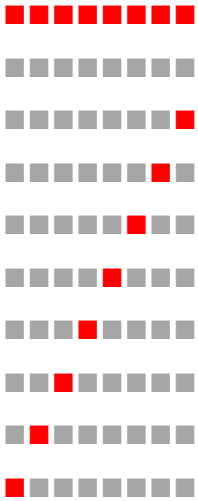
Finalmente, en la función *Receive_Data_RX_Mode_nRF24L01* se reafirma la habilitación de *checksum*, se marcan 10 segundos como tiempo máximo de espera del dispositivo hasta saltar el IRQ ERROR, se vuelve a indicar el ancho del *payload*, y por último, se indica en que tabla de bytes se almacenará el mensaje de llegada.

```
                if(informe & 0b00010000)
                {
                        LATB=0b00111100;
                }
                else if(!(informe & 0b00010000))
                {
                        if(informe & 0x01)
                        {
                                LATB=0b11111111;
                                Delay10KTCYx(100);
                                LATB=informe>>1;
                                Delay10KTCYx(100);
                        }
                        else if(!(informe & 0x01))
                        {
                                LATB=0b01010101;
                                Delay10KTCYx(100);
                                LATB=informe>>1;
                                Delay10KTCYx(100);
                        }
                        LATB=mensaje_llegada[0];      Delay10KTCYx(50);
                        LATB=mensaje_llegada[1];      Delay10KTCYx(50);
                        LATB=mensaje_llegada[2];      Delay10KTCYx(50);
                        LATB=mensaje_llegada[3];      Delay10KTCYx(50);
                        LATB=mensaje_llegada[4];      Delay10KTCYx(50);
                        LATB=mensaje_llegada[5];      Delay10KTCYx(50);
                        LATB=mensaje_llegada[6];      Delay10KTCYx(50);
                        LATB=mensaje_llegada[7];      Delay10KTCYx(50);
                }
                while(PORTDbits.RD1);
        }
}
```

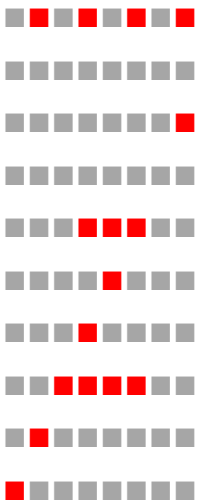
Por último se obtiene la información del proceso y se muestra al usuario mediante un código de visualización en los LEDs. El programa finaliza del mismo modo con la condición

de que el programa sólo se reinicie en el caso de que el usuario accione el pulsador S0. Los posibles resultados y su código de visualización son:

- **Recepción correcta del paquete (*checksum* válido)**: se encienden los ocho LEDs durante un segundo y se indica posteriormente el *pipe* de procedencia del paquete durante otro segundo. Seguidamente se muestra cada byte del *payload* con medio segundo entre cada visualización para poder verificar el contenido del paquete recibido. En este caso, la visualización correspondiente sería:



- **Recepción incorrecta del paquete (*checksum* inválido)**: se encienden LEDs alternados durante un segundo y se indica posteriormente el *pipe* de procedencia del paquete durante otro segundo. Seguidamente se muestra cada byte del *payload* con medio segundo entre cada visualización para poder verificar el contenido del paquete recibido y observar si efectivamente ha habido un error en la transmisión vía RF. Una posible visualización para este caso sería:



- **Error de interrupción IRQ**: se encienden los cuatro LEDs del medio (no se analizan los 4 bits del *pipe* de llegada).

Sobre el código del dispositivo receptor sólo cabe destacar que, si se quisiese establecer una red dispositivos conectados entre sí donde un dispositivo recibiese información de diferentes emisores, la opción más adecuada sería utilizar otros *pipes* de llegada de datos. Concretamente, se debería reservar un espacio de la memoria RX FIFO y una dirección de RF para cada *pipe* y finalmente ejecutar la función *Receive_Data* tantas veces como *pipes* se vayan a utilizar.

10.3. Conclusión del test de validación

Para una primera validación de la librería elaborada y del sistema de comunicación por RF en general se han programado un dispositivo emisor y un dispositivo receptor con el código anterior (sólo se ha cambiado la línea `while(PORTDbits.RD1);` por un retardo de tres segundos para que el ciclo de envío/recepción se repita una y otra vez sin tener el usuario que hacer uso del pulsador S0).

Ambas placas se han encendido y colocado a una distancia de tres metros; y para obtener una estimación de la fiabilidad del proceso, se ha observado su comportamiento durante **100 ciclos de comunicación**:

- En **89** ocasiones el proceso **se ha realizado satisfactoriamente**: el emisor informa de un envío correcto y el receptor recibe el paquete correctamente (*checksum* válido y *payload* visualizado correcto).
- En **5** ocasiones se ha producido **un error de *checksum***: el emisor informa de un envío correcto pero el receptor alerta del fallo en el *checksum*, y efectivamente el *payload* es incorrecto.
- En **1** ocasión **se alerta en el receptor un error de *checksum* pero el *payload* mostrado es correcto** (ha habido un error en el último byte del paquete, que es el *checksum* del dispositivo TX).
- En **1** ocasión **se confirma un *checksum* válido en el receptor pero el *payload* mostrado es incorrecto** (se ha distorsionado el *payload* transmitido pero la suma de sus bytes cumple el *checksum*).

- En **4** ocasiones no se ha producido comunicación porque en se ha dado **un error de IRQ** en transmisor o receptor.
- En ninguna ocasión el transmisor confirma un envío pero el receptor no recibe información.
- En ninguna ocasión el transmisor comunica el fallo del envío pero el receptor ha recibido un paquete de datos que muestra.

Se deduce por lo tanto de estos resultados que el proceso de comunicación sigue, en la mayor parte de las ocasiones, un comportamiento previsto (un 98%). Concretamente **sólo falla el sistema desarrollado en dos ocasiones** (un 2%), esto es, cuando se afirma que ha existido un error de *checksum* y el *payload* es correcto y cuando se afirma un *checksum* válido pero el *payload* es incorrecto. En estos casos se produce un error en el paquete enviado pero éste no es detectado por el sistema de verificación, que para ser más fiable, debería incluir más bits redundantes (uso por ejemplo de bytes de CRC).

Las ocasiones en las que hay un fallo del *checksum* o de la interrupción IRQ (que representarían solamente un 9% del los casos mientras que en el 89% el envío es correcto) corresponden a **situaciones contempladas dentro de los posibles comportamientos del sistema**, que dentro del código del estudiante deberían comportar un reinicio de la comunicación RF. Son situaciones no preocupantes (no suponen un error desconocido) mientras se den en escasas ocasiones para evitar que el proceso sea excesivamente lento.

Por otro lado, no se da en ningún momento la situación de que el transmisor confirme un envío y el receptor no lo reciba o que el transmisor comunique un fallo y el receptor haya recibido un paquete de datos, **situaciones que sí se producían antes** de crear el bucle `while(!IRQ)` en la función *Check_Data_Sent*. Tampoco se produce nunca un bloqueo del sistema en el bucle de espera de interrupción `while(IRQ)` gracias a la creación de los contadores y su periodo máximo de estancia en el bucle.

Es importante destacar que, para estimar la fiabilidad del sistema con una mayor precisión, se debería repetir este experimento hasta obtener unos márgenes de confianza suficientes, pero se estima que es **un resultado suficiente para dar unos resultados orientativos**.

En definitiva, tanto el funcionamiento de las placas programadas como del sistema de RF en general, puede ser **considerado válido**. No solamente es un sistema fiable sino también un sistema rápido y efectivo. Tanto su inicialización como su ejecución y monitorización son sencillas y completas; y el sistema de visualización de LEDs, aunque algo precario, permite conocer todos los aspectos de la comunicación por RF.

Como conclusión de este apartado y preámbulo del siguiente, sólo cabe destacar que esta experimentación supone sólo una primera confirmación del correcto funcionamiento de la comunicación por RF mediante los módulos nRF24L01; pues no se olvide que este proyecto se orienta a la realización de prácticas por parte de varios estudiantes trabajando con varias placas a la vez en un amplio espacio de trabajo. El funcionamiento bajo estas condiciones de trabajo del sistema elaborado definirá sus prestaciones y limitaciones finales, y en definitiva, la adecuación de este proyecto a su objetivo final.

11. Prestaciones y limitaciones del módulo RF

Con el objetivo de validar este proyecto de forma definitiva se deben evaluar las prestaciones y las limitaciones de la placa Open18f4520 y su módulo nRF24L01 en las mismas condiciones en las que será utilizada, esto es, en un laboratorio de electrónica donde diversos alumnos realizarán sus prácticas de Radio Frecuencia. **Las condiciones de trabajo** más destacables serán:

- **Las distancias** entre emisor y receptor podrán ser superiores a la del experimento anterior de 3 metros, sin embargo, se considera que en ningún caso ahora se superarán los **10 metros**.
- Si se utilizan otros dispositivos en el laboratorio, la comunicación por RF podría verse perturbada impidiendo la comunicación entre emisor y receptor de forma parcial o total. Se deben contemplar las **posibles interferencias** dentro del espacio de trabajo.
- En el laboratorio podrán funcionar **diversas placas a la vez** (hasta 10 placas), algunas en configuración como transmisor y otras como receptor. Se deberá asignar a cada pareja TX y RX una dirección y una frecuencia de trabajo; pero es necesario comprobar que cada pareja funciona correctamente y no se ve afectada por el resto de placas.
- Se valora también analizar si algún tipo de material u objeto puede ejercer de **barrera entre emisor y receptor** impidiendo el paso de las ondas de RF.
- Se podrá establecer y verificar **una comunicación en estrella** (varios transmisores envían a un receptor o varios receptores reciben un mismo envío de un mismo transistor) o **una pequeña red de comunicación**.

En definitiva, el resultado de este estudio determinará finalmente si el trabajo elaborado es apto para su aplicación práctica dentro de la docencia de microcontroladores y Radio Frecuencia. Este estudio y los resultados finales obtenidos serán explicados y valorados en la defensa de este proyecto.

12. Presupuesto económico

MATERIAL Y SOFTWATE			
ITEM	**Cantidad**	**Precio Unidad (€/u)**	**Coste Total Item (€)**
Pack Open18F4520 Package B (Placa, PIC18F4520, módulo nRF24L01, cables de conexión, periféricos)	2	73,75	147,5
Programador PICkit 3 (Programador, cable USB)	1	54,8	54,8
PIC18F4520	1	8,63	8,63
Protoboard	1	5,08	5,08
Bobina de cable	1	1,81	1,81
Pines de conexión (machos)	30	0,06	1,8
Pines de conexión (hembras)	20	0,06	1,2
Software MPLAB y compilador C18 (Software libre)	1	0	0
		TOTAL	219,02

TIEMPO DE TRABAJO			
	Horas	**Coste (€/h)**	**Coste Total (€)**
Trabajo dedicado al estudio previo	100	10	1000
Trabajo dedicado al diseño de la librería	300	10	3000
Trabajo dedicado a la prueba y validación del proyecto	150	10	1500
		TOTAL	5500

CONSUMO DE RECURSOS			
	Horas	**Coste (€/KWh)**	**Coste Total (€)**
Consumo electricidad del PC y placas (100 W)	500	0,14	7
Consumo electricidad del espacio de trabajo (100 W)	550	0,14	7,7
		TOTAL	14,7

COSTE TOTAL DEL PROYECTO	
	Coste (€)
Material y software	219,02
Tiempo de trabajo	5500
Consumo de recursos	14,7
TOTAL	**5733,72**

ETSEIB

13. Impacto medioambiental

Para evaluar el impacto medioambiental de este proyecto antes de su aplicación práctica es necesario recordar su objetivo último: intercomunicar en un mismo laboratorio de prácticas de electrónica las diferentes placas de los alumnos para facilitar así el aprendizaje de microcontroladores y la comunicación por RF.

Esta aplicación práctica del proyecto motiva que en esta fase del estudio no solamente se trate el hardware utilizado (la placa Open18f4520 y su módulo RF) y su impacto en el medio ambiente como futuro residuo o producto reciclable, sino que también se valore el uso de la RF y su incidencia sobre personas, demás seres vivos y objetos. Concretamente es necesario valorar la incidencia de la RF en los alumnos y en otros dispositivos electrónicos del laboratorio a las distancias que se encontraran las antenas de las placas y a la potencia generada por ellas.

Respecto al **impacto medioambiental causado por el hardware como futuro residuo**, una vez termina la vida útil del producto que debe ser lo más larga posible, las opciones que pueden ser consideradas son el reciclaje o su desecho en gestores de residuos especializados en residuos electrónicos.

El reciclaje de componentes a nivel particular es relativamente inviable, ya que algunos son de tipo SMD (*tecnología de montaje superficial*) y la difícil reutilización de otros supondría un coste mayor que adquirir nuevos componentes por precios bajos. Es en este caso más asequible, si se da la situación, un reciclado de la placa en su conjunto dándole nuevos usos.

En último lugar, a pesar de ciertas fases de reciclado, la placa deberá ser desechada de la forma correcta, esto es, en **centros de gestión de residuos de aparatos eléctricos y electrónicos (RAEE)**. Su desecho en contenedores de basura común supondría su acumulación en vertederos donde el efecto de los materiales contaminantes de la placa (como el PCB, TBBA, PBB, PVC,etc) resultaría perjudicial para el medio ambiente. En los centros de gestión de residuos se lleva a cabo un reciclaje efectivo de los componentes, se desechan de forma apropiada los elementos contaminantes y finalmente los productos no nocivos son acumulados.

En lo que al uso de las placas se refiere, debe considerarse **el impacto medioambiental de la comunicación por ondas de RF** dentro de un aula con diversos alumnos y otros dispositivos electrónicos; y para caracterizarla correctamente, deben definirse de forma concreta dos parámetros: la frecuencia electromagnética de trabajo y la potencia generada por las antenas.

La **frecuencia de trabajo** del nRF24L01 es de entre 2.4 y 2.5 GHz (enmarcada dentro de la Alta Frecuencia de entre 3 MHz y 3 GHz; y esta a su vez, enmarcada dentro del espectro de radiofrecuencia situada entre los 3Hz y los 300 GHz de todo el espectro electromagnético). Estos espectros de frecuencia son también los utilizados, por ejemplo, en procesos médicos que buscan la regeneración de tejidos o el calentamiento de ciertas zonas del cuerpo para favorecer su relajación o recuperación; y por otro lado, a potencias mucho más altas, estos espectros pueden causar en un individuo dolores, quemaduras, pérdidas de visión, o muchos otros daños ya alertados por la OMS. La diferencia entre unos casos y otros reside en dos factores adicionales, la potencia del campo electromagnético aplicado y el tiempo de exposición.

La **potencia de salida del nRF24L01**, la potencia generada en el campo electromagnético, corresponde como máximo a 0dBm, lo que corresponde a 1mW según la fórmula siguiente:

$$P_{dBm} = 10 \cdot \log_{10} P_{mW}$$

Para determinar por tanto la incidencia del campo generado por el nRF24L01 en el organismo, se tiene en consideración el parámetro **SAR** (*specific absortion rate*), que es una medida de la potencia máxima con que un campo electromagnético de radiofrecuencia es absorbido por un tejido. Éste criterio se emplea en campos de frecuencia de entre 100 KHz y 100 GHz y a distancias de la antena que pueden ser tan bajas como 20cm o menos, de modo que es perfectamente aplicable al dispositivo en cuestión (fuera de este rango se debe trabajar mediante criterios de límite de densidad de potencia).

Según la Organización Mundial de la Salud, dentro del rango de frecuencias de entre 1 MHz y 10 GHz, en necesario un SAR de por lo menos **4W/Kg** para producir efectos adversos en la salud de gravedad; y según al *Safety Code 6* del departamento de sanidad de Canadá, para un largo periodo de exposición (por ejemplo de un operario que trabaja durante horas), no es admisible una densidad de potencia mayor de **0.4W/Kg** para evitar cualquier tipo de lesión. Tomando este último valor en consideración y suponiendo que toda la potencia de RF generada por el nRF24L01 fuera absorbida por un estudiante (de 60 Kg), la potencia máxima admisible sería de 24W, muy superior en definitiva a la potencia real generada de 1mW.

En conclusión, **el trabajo con el nRF24L01 resulta inocuo para el estudiante** dada la bajísima potencia del campo RF generado y, además, el corto periodo de exposición (que no sería ni siquiera la duración de la sesión de prácticas al estar la librería implementada de forma que la antena, al suponer un mayor coste energético, sólo se active en tiempos muy cortos: para la transmisión, como mucho durante 70ms por envío; y para la recepción, como mucho durante 25 segundos).

Conclusiones

Tras las fases de diseño de la librería y de verificación del sistema de comunicaciones por RF, se concluye finalmente que el trabajo desarrollado responde de forma satisfactoria a los objetivos propuestos para este proyecto.

En primer lugar, se ha conseguido establecer una configuración en el módulo de RF nRF24L01 que permite sacar el máximo rendimiento de sus prestaciones y, a la vez, que todas ellas sean configurables. Mediante no más de dos funciones es perfectamente realizable el envío o recepción de un paquete de datos lo más rápido que permite el nRF24L01. Además, el sistema es perfectamente controlable gracias a la información que retornan las funciones de envío y recepción de datos.

Por otro lado, se han puesto solución a los tres grandes problemas o errores de funcionamiento del nRF24L01: mediante la creación de procesos de verificación se pone remedio a los fallos en la escritura y lectura de registros en el nRF24L01, y mediante el establecimiento de contadores se evita el bloqueo del módulo en la espera de la interrupción IRQ. También la creación de la verificación por *checksum* hace el sistema más fiable e ilustrativo para los alumnos. En definitiva, mediante estas mejoras de los primeros diseños, en ningún momento se produce algún error inesperado o se pierde el control sobre el proceso

En conclusión, dada la alta fiabilidad del sistema de comunicación y su alta tasa de envíos/recepciones de paquetes correctos, el sistema resulta apto para el aprendizaje de microcontroladores y comunicaciones por RF. Es un sistema sencillo de configurar y ejecutar por parte de los estudiantes, que ofrece por otro lado resultados variados y didácticos.

Como recomendación final simplemente cabe decir que para comunicaciones entre microcontroladores realmente rápidas y efectivas a mayores distancias, el módulo *nRF24L01 RF Board* no sería el más indicado dados sus errores de funcionamiento y su baja potencia. De hecho, las mejoras desarrolladas en el proyecto, simplemente ponen remedio a estos fallos a costa de un tiempo extra de ejecución; como el que se produce al verificar y reescribir registros o como el de declarar un error de interrupción y tener que reiniciar el proceso. Para el campo de la docencia, este hecho es incluso más didáctico para el alumno, pero se debería descartar su aplicación en un campo industrial donde la velocidad de comunicación es primordial.

Bibliografía

MARTÍN CUENCA, E [et al.]. *Microcontroladores PIC. La clave del diseño.* Madrid, Thomson, 2003.

MICROCHIP TECHNOLOGY INC. *MPLAB C18 C Compiler User's Guide*. U.S.A., 2005.

MICROCHIP TECHNOLOGY INC. *PIC18F4520 DATA SHEET*. U.S.A., 2008.

MICROCHIP TECHNOLOGY INC. *MPLAB IDE User's Guide*. U.S.A., 2009.

NORDIC SEMICONDUCTOR ASA. nRF24L01 DATA SHEET. Norway, 2005.

WAVESHARE ELECTRONICS [http://www.wvshare.com]: Información de la placa de desarrollo Open18f4520 y del módulo nRF24L01 RF *Board*.

MICROCHIP [http://ww1.microchip.com/downloads/en/DeviceDoc/spi.pdf]: Tutorial sobre el SPI de *Microchip*.

TUTORIALES PIC [http://picfernalia.blogspot.com.es/2013/04/comunicaciones-serie-spi.html]: Teoría sobre la comunicación SPI en los microcontroladores PIC.

ORGANIZACIÓN MUNDIAL DE LA SALUD [http://www.who.int/peh-emf/publications/facts/fs226/es/]: Campos electromagnéticos y salud. Impacto medioambiental de la RF en seres humanos.

HEALTH CANADA, SAFETY CODE 6 [http://www.etsist.upm.es/estaticos/catedra-coitt/web_salud_medioamb/Informes/informes_PDF/rf/HealthCanada/99ehd237.pdf]:*Limits of Human Exposure to Radiofrequency Electromagnetic Fields in the Frequency Range from 3 kHz to 300 GHz*.

MINISTERIO DE AGRICULTURA, ALIMENTACIÓN Y MEDIOAMBIENTE [http://www.magrama.gob.es/es/calidad-y-evaluacion-ambiental/temas/prevencion-y-gestion-residuos/flujos/aparatos-electr/]: Impacto ambiental de aparatos eléctricos y electrónicos.

Referencias bibliográficas de las figuras

WAVESHARE ELECTRONICS [http://www.wvshare.com/]:

Figuras 3-1, 4-1 y 4-2.

MICROCHIP TECHNOLOGY INC. *PIC18F4520 DATA SHEET*:

Figuras 3-4, 3-5, 3-6 y 3-7.

NORDIC SEMICONDUCTOR ASA. nRF24L01 DATA SHEET:

Figuras: 4-3, 4-4, 4-5, 4-6, 6-1, 6-2 y 6-3.

Tutoriales PIC [http://picfernalia.blogspot.com.es/2013/04/comunicaciones-serie-spi.html]:

Figura: 3-3.

Serial Peripheral Interface [http://es.wikipedia.org/wiki/Serial_Peripheral_Interface]:

Figura: 3-2.